POK User Guide

POK Team

December 2, 2023

Contents

List of Figures

Chapter 1

Introduction

1.1 What is POK?

POK is a kernel dedicated to real-time embedded systems. This kernel works on various architectures. The configuration code, the deployment code as well as the application (userland) code can be automatically generated, achieving zero-coding approach.

One main goal of POK is to be compliant with many industrial standards. In the embedded domain, many API exist for different application domains (avionics, railway, automotive). However, concepts remain the same. POK proposes a canonical adaptive kernel compliant with several industrial standards.

1.2 Purpose of this document

This document provides general information about POK. It also present available API, describes them and detail how to configure the kernel.

1.3 Supported platforms

At this time, POK supports the following platforms:

- x86, emulation with QEMU
- PowerPC
- Leon3, a platform for aerospace applications

1.3.1 x86

The x86 support is included to rapidly develop applications and test them into an emulator like QEMU or bochs.

1.3.2 PowerPC

A PowerPC port is available. The port is available for the following BSP:

- prep

1.3.3 LEON3

A port for the LEON3 architecture (a typical architecture in the aerospace domain) is currrently in progress. Please contact the POK team if you are interested by this port.

1.4 Supported standards

POK is compliant with the following standards:

- ARINC653
- MILS

To achieve standard compliance, POK relies on a minimal API that provides few canonical services. These services then interact with the kernel to interact with other nodes/processes.

1.4.1 ARINC653 support

At this time, POK is compliant with the ARINC653 standard, meaning that it provides partitioning functionnalities. On the userland-side, it provides all the C and Ada API of the first part of ARINC653.

However, POK does not have an XML parser to automatically create the configuration/deployment code from ARINC653 XML files.

1.4.2 MILS

MILS stands for Multiple Independent Level of Security. POK, by defining strong partitioning, can be MILS-compliant, depending on its configuration. Most of the time, the MILS compliant can be reached with a analysis of the system in terms of security. To encrypt data, POK relies on the OpenSSL library, released under a BSD-license (see http://www.openssl.org for more information).

1.5 About the POK team

POK is a research project. It was made to experiment partitioned architectures and build safe and secure embedded systems. It was initiated during a PhD thesis at TELECOM ParisTech[1] and LIP6[2] laboratories. The developer leader is JULIEN DELANGE, who did his PhD on partitioned architectures.

[1]http://www.telecom-paristech.fr
[2]http://www.lip6.fr

However, several students from the EPITA school[3] joined the project and improve for several purposes (projects, exercises, fun, . . .). In addition, other people contributed to the project for several reasons.

There is a list of the people involved in the project (alphabetical order):

- Fabien Chouteau (LEON port)
- Tristan Gingold (PowerPC port)
- François Goudal (initial version known as the Gunther project)
- Laurent Lec (Realtek 8029 driver, kernel IPC and so on)
- Pierre-Olivier Haye (memory protection)
- Julian Pidancer (initial work on partition isolation)

Hope the team will grow up in a near future !

[3]http://www.epita.fr

Chapter 2

Installation

2.1 Supported development platforms

- Linux
- Mac OS X
- Windows

2.2 Get more information

The following information are the standard procedures. It may be out of date or miss something. In that case, you will find updated information on the POK website (`http://pok.gunnm.org`) and its wiki section.

In addition, there are some tutorials and information about the installation of required tools.

2.3 Linux/MacOS

2.3.1 Pre-requires

- The GNU-C Compiler (aka GCC), version 3.x or 4.x
- GNU binutils
- GNU Zip (aka gzip)
- Mtools (MS-DOS disk utilities)
- AWK
- Perl (with `XML::XPath::XMLParser` and `XML::LibXML` modules)

- QEMU (for x86 emulation)
- *Ocarina* (for code generation only)
- TSIM (for Leon3 emulation)

Note for MacOS users

POK uses the ELF format to store partitions. Unfortunately, this binary format is not supported by Mac OS X tools. To use POK, you must use a development toolchain that supports the ELF format.

For that, you can easily build an ELF cross compiler using MacPorts. The name of the required packages are `i386-elf-gcc` and `i386-elf-binutils`.

Moreover, Mac OS X does not provide necessary Perl modules but you can install them with MacPorts. The package names are `p5-xml-xpath` and `p5-xml-libxml`.

2.3.2 Running POK

Ocarina is needed by POK. A script is provided to automatically install the latest build:

```
$ sh ./misc/get_ocarina.sh
```

You can then try to build and run some of the POK examples located in the 'example' directory.

```
$ cd examples/partitions-threads
$ make
$ make -C generated-code run
```

A whole chapter of this documentation is dedicated to those examples and their purpose.

2.4 Windows

2.4.1 Pre-requires

There is many pre-requires to run POK on Windows. To make a better user experience, we provide cross-development tools to build and use POK.

For cross-development tools can be retrieved from the website of the project. Then, unzip the tools in a directory called `crosstools`, at the root directory of the project.

Once you have this directory, run the file `configure.bat` located in this directory. Is everything is wrong, a warning will be displayed on the screen.

For code generation, you can install *Ocarina* for Windows. All installation instructions are available on Ocarina website.

Chapter 3

Getting started

3.1 First experience with POK

To build and run your first system with POK, you must have the *Ocarina* code generator[1] and all the software required by POK (a list is available in the chapter 2).

Then, perform the following actions :

1. Issue `make configure` at the top directory of POK. Is something is missing, install it !
2. Enter the directory `examples/partitions-threads` by typing this command in a terminal:

   ```
   cd examples/partitions-threads
   ```

3. Invoke `make`. It will generate configuration and application code with *Ocarina*.
4. Invoke `make run` in the `generated-code` directory. You can do that with the following commands

   ```
   make -C generated-code run
   ```

   Using this command, *qemu* is launched and your system is being executed.

Now, the next sections will explain how to configure the kernel and the partition layer for your own projects.

[1] Available at http://aadl.telecom-paristech.fr

3.2 Development cycle

POK has a dedicated development cycle which avoid the compilation of the kernel. The development process automatically compiles the kernel, the partitions with the application code and assemble them into a bootable binary (you can see the illustration of this development process in figure 7.2.

Due to the tedious configuration efforts of each layer, a tool that automatically configures the kernel and the partitions from AADL descriptions is available (the *Ocarina* code generator). You can also configure each part by yourself by writing C configuration code.

3.3 Configure POK: the `conf-env.pl` script

POK distribution can be configured so reach different goals. The basic configuration is automatically performed. However, in some cases, you want to use some additional options.

At first, the configuration of POK is made with the `conf-env.pl` script, located in the `misc` directory. So, issue `./misc/conf-env.pl` to use the default configuration. The configuration is automatically produced by this script and written in the `misc/mk/config.mk` file.

Then, the `conf-env.pl` script can be used with additional switches to enable some options of POK. There is a list of these switches:

- `--help` : print help menu
- `--with-xcov` : use xcov from the coverage[2] project. With this option, when you invoke `make run` after building a system, the emulator will be stopped after 40 seconds of execution and analyses the code coverage of the system.
- `--with-floppy` : add an additional rule so that you can automatically install the POK binary into a bootable floppy. In consequence, you can invoke `make install` in the generated directory to create this floppy disk image.
- `--with-instrumentation` : automatically insrument kernel and partition code and produce additional output to trace system activity. This functionnality produces additional files to trace and analze POK behavior with third-party tools such as Cheddar10.2.

3.4 Automatic and manual configuration

The automatic code generation finely configure the kernel and enable only required functionnalities. If is especially efficient for embedded systems when you have to avoid useless features and reduce the memory footprint. In addition, it avoids all potential errors introduced by the code produced by human developers. The automatic configuration process is detailed in chapter 4.

[2]see `http://forge.open-do.org/projects/couverture/`

On the other hand, you can also configure the kernel and the partitions by yourself. In this case, the configuration will be very difficult since POK has many configuration directives. This configuration process is detailed in the next section.

3.5 Kernel configuration with ARINC653 XML files

You can also configure the kernel with an ARINC653 XML file. The tool is available in POK releases in the `misc/` directory. More information can be found in section 5.1.

3.6 How to write my manual code ?

At this time, if you try to write the configuration code by yourself, you have to read the configuration directives of POK. The fact is that you need to write the configuration code by yourself and make your own build system that supports POK (the automatic configuration process output code and automatically create the build system for you).

In that case, the best is to start from a working example. Try to take the generated code from the `examples` directory. It could be efficient since they are many examples that use various services of the runtime.

Finally, the POK team plans to release a tool that would help the developper in the configuration of the kernel and partitions. Such a tool would be graphic (like the well-known `make menuconfig` of the Linux kernel) and would propose to configure kernel and partitions.

3.7 Using Ada for partitions

Both C and Ada can be used for partitions. Ada will nevertheless require some tuning to run into POK, only a GCC toolchain that handles Ada is needed.

Since POK partitions are loaded by executing their main function, one of the Ada packages must export a function as `main`. Moreover, the runtime should be disabled using `pragma No_Run_Time`.

The following piece of code is an example of how to proceed:

```
-- main.ads
pragma No_Run_Time;
with Interfaces.C;

package Main is
   procedure Main;
   pragma Export (C, Main, "main");
end Main;


-- main.adb
package body Main is
   procedure Printf (String : in Interfaces.C.char_array);
   pragma Import (C, Printf, "printf");

   procedure Main is
   begin
      Printf ("Hello world!");
   end Main;
end Main;
```

An ARINC653 layer is also available in `libpok/ada/arinc653` and should be used the same way as described above.

3.8 Run POK on Leon3

To build and run POK on Leon3, you must have the TSIM simulator[3].

Then, perform the following actions :

1. Add tsim-leon3 directory to you PATH environment variable.
2. Issue `make configure` at the top directory of POK.
3. Enter the directory `examples/partitions-scheduling` by typing this command in a terminal:

   ```
   cd examples/partitions-scheduling
   ```

4. Invoke `make ARCH=sparc BSP=leon3`. It will generate configuration and application code with *Ocarina*.
5. Invoke `make ARCH=sparc BSP=leon3 run`. Using this command, *TSIM* is launched and your system is being executed.

[3]Evaluation version available at ftp://ftp.gaisler.com/gaisler.com/tsim/

Chapter 4

Automatic configuration and configuration with AADL models

4.1 Proposed development process

Using AADL models can help system designers and developers in the implementation of partitioned architectures. POK can be configured automatically using AADL models. In fact, the AADL is very efficient for the design of real-time embedded systems: designers specify their architecture with respect to their specificites and requirements. Then, the *Ocarina* toolsuite analyzes the architecture and automatically generates code for POK.

The code generation process automatically configures the kernel and the partitions. The developpers should provide the application-level code. This application-level code can be traditional code (Ada, C) or application models (Simulink, Scade, etc.).

Our code generator was integrated in the *Ocarina* AADL toolsuite. It is a popular toolsuite for AADL models handling. It provides several functionnalities, such as models analysis, verification and code generation. In the context of POK, we rely on these functionnalities to verify and automatically implement the system.

The development process is illustrated in the figure 4.1: the developper provides AADL models, the code generator creates code that configures kernel and libpok layers. Compilation and integration is automatically achieved by the toolchain and creates final binary runnable on embedded hardware.

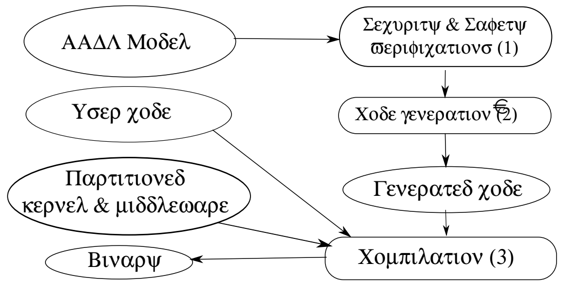

Figure 4.1: Model-Based development process

4.2 Use the pok toolchain for model analysis, validation, code generation, compilation and execution (the `pok-toolchain.pl` script)

We provide a toolchain that provides the following functionnalities:

1. **Model analysis**: check that your AADL model is correct.
2. **Model validation**: validate the requirements specified in the model
3. **Code generation**: automatically generate the code for its execution with POK
4. **Compilation**: automatically compile and create binaries

4.2.1 Use the `pok-toolchain.pl` script

The toolchain is implemented in a script called `pok-toolchain.pl`. This script is used to perform the different actions of the development process. This script has the following options:

- `models=` is a **REQUIRED** option. It specifies the AADL models you use for this system. For example, you can specify `models=model1.aadl,model2.aadl`. This is the list of your models.
- `nogenerate`: do not generate the code. By default, the toolchain generates the code from AADL models.
- `norun`: do not run the generated systems. By default, the toolchain generates code and run generated systems.
- `nocheck`: do not validate the architecture.
- `root=system_name`: Specify the root system of your architecture. If your models contain several `system` components, you need to specify what is the AADL root `system` component.
- `arinc653`: use ARINC653 code generation patterns.

4.2.2 Example of use

The following line will generate ARINC653-compliant code from `model1.aadl` model.

```
pok-toolchain.pl --models=model1.aadl --arinc653
```

The following line will generate code and compile it, but will not run generated system.

```
pok-toolchain.pl --models=model1.aadl --no-run
```

4.3 Model validation

Our toolchain automatically validates models requirements before generate code. It was made to help system designer in the verification of its architecture.

Our validation process is based on *Ocarina* and the REAL language, which is a constraint language for the AADL. Its quite similar than OCL language (designed for UML), except that is specific to AADL and thus, makes easier the validation of AADL model. You can have additional information about *Ocarina* and REAL on *http://www.aadl.telecom-paristech.fr*. With REAL, the user defines one or several *theorems* that express what we want to check.

There is a list of the theorems used in the POK toolchain and what we verify:

1. **MILS requirements enforcements**: we check that each partition has one security level and connected partitions share the same security levels. For that, the underlying runtime and the connections should support appropriate security levels.

2. **Bell-Lapadula and Biba security policies**: for connected partitions, we check the Bell-Lapadula and Biba security policies (no read-up/write-down, . . .). With that, we ensure that the architecture is compliant with strict security guidelines.

3. **Memory requirements**: we check that required size by a partition is less important than the size of its bounded memory component. In other words, we check that the memory segment can store the content of the partition. We also check that the requirements described on partitions are correct regarding their content (threads, `subprograms` size, . . .).

4. **Scheduling requirements (Major Time Frame)**: for each `processor` component, we check that the major time frame is equal to the sum of partitions slots. We also check that each partition has at least one time frame to execute their threads.

5. **Architecture correctness**: we check that models contain memory components with the appropriate properties. We also check that `process` components are bound to `virtual processor` components.

4.4 POK properties for the AADL

The AADL can use user-defined property sets to add specific properties on AADL components. On our side, we define our own AADL properties, added to AADL components to describe some specific behavior.

The POK property set for the AADL can be found in the annex section.

In addition, POK and its associated AADL toolsuite (*Ocarina*) supports the ARINC653 annex of the AADL. So, you can use models that enforces the ARINC653 annex with POK. The ARINC653 property set for the AADL is included in the annex section of this document.

4.5 Modeling patterns

This section describes the code generation patterns used to generate code. So, it explain the mapping between AADL models and generated code. To understand this section, you have to know the AADL. You can find tutorials in the internet about this modeling language (Wikipedia can be a good starting point).

4.5.1 Kernel

The kernel is mapped with the AADL `processor` component. If the architecture is a partitioned architecture, it contains partitions runtime (AADL `virtual processor` components).

Scheduling

The scheduling requirements are specified in `process` components properties. The `POK::Slots` and `POK::Slots_Allocation` properties indicate the different time slots for partitions execution (in case of a partitioned architecture).

In addition, the `POK::Scheduler` is used to describe the scheduler of the processor. If we implement an ARINC653 architecture, the scheduler would be static.

4.5.2 Device drivers

In POK, device drivers are executed in partitions. It means that you must embedds your code in partitions and drivers are isolated in terms of time and space. Consequently, drivers rely on the kernel to gain access to hardware resources (I/O, DMA and so on).

To do that, AADL components are considered as partitions. So, when your model contains an AADL device, the underlying code generator consider that it is a partition. So, you have to **associate** device components with `virtual processor` components to indicate the partition runtime of your driver.

However, the device driver cannot describe the actual implementation of the driver. For that, we use the `Implemented_As` property. This property points to an abstract component that contains the implementation of our driver. Annexes of the current document provide an example of the modeling of a driver (see section): the `driver_rtl8029` `abstract` component models the driver by defining a process that contains threads. These threads handle the device and perform function calls to access to hardware resources.

However, for each device, POK must know which device you are actually using. So, you have to specify the `POK::Device_Name` property. It is just a string that indicate which device driver you are using with this device component.

In addition, for network device that represent ethernet buses, you must specify the hardware address (also known as the MAC address). For that, we have the property `POK::Hw_Addr`. This property must be associated with a device component.

Supported device drivers

At this time, we only support one device driver : the realtek 8029 ethernet controller. This device is simulated by QEMU and thus, can be easily tested and simulated on every computer that uses QEMU.

However, implementing other device driver can be easily achieved, by changing the `Device_Name` property in the model and adding some functions in the `libpok` layer of POK.

Case study that defines a device driver

You can find an example of an implementation of a device driver in the `examples/network` directory of each POK release. It defines two ARINC653 module that communicate across an ethernet network. Each module contains one partition that communicate over the network. You can have more information by browsing the `examples/network` directory.

4.5.3 Partitions

In case of a partitioned architecture, we need to describe partitions in your AADL model. In that case, partitions are mapped with two AADL components: `process` and `virtual processor`.

The `virtual processor` models the runtime of the partition (its scheduler, needed functionalities and so on).

We associate the `virtual processor` component (partition runtime) and its `process` component (partition address space) with the `Actual_Processor_Binding` property.

Scheduling

The scheduling policy of the partition is specified with the `POK::Scheduler` property in the `virtual processor` component (partition runtime).

Memory requirements

You can specify the memory requirements in two ways.

First, with the `POK::Needed_Memory_Size` property on the process (partition address space). It will indicate the needed memory size for the process.

You can also specify memory requirements with AADL `memory` components. You bind a `memory` component to a partition process component with the `Actual_Memory_Binding` property. In that case, the properties (`Word_Size`, `Word_Count`, `...`) of the memory component will be used to generate its address space.

Additional features

You can specify which features are needed inside the partition (libc, libmath and so on). In that case, you have to specify them with the `POK::Additional_Features` property.

4.5.4 Threads (ARINC653 processes)

Threads are contained in a partition. Thus, these components are contained in a `process` component (which models a partition).

There is the supported properties for threads declaration:

- `Source_Stack_Size`: the stack size of the thread
- `Period`: the actual period of the thread (execution rate)
- `Deadline`: the actual deadline of the thread (when the job should finish)
- `Compute_Exeution_Time`: the execution time needed to execute the application code of the threads.

4.5.5 Inter-partitions channels

Queuing ports

Queuing ports are mapped using AADL event data ports connected between AADL processes. This ports are also connected to `thread` components to send/receive data.

Sampling ports

Queuing ports are mapped using AADL data ports connected between AADL processes. This ports are also connected to `thread` components to send/receive data.

4.5.6 Intra-partitions channels

Buffers

Buffers are mapped using AADL event data ports connected between AADL threads. This ports must not be connected outside the process.

Blackboards

Buffers are mapped using AADL data ports connected between AADL threads. This ports must not be connected outside the process.

Events

Buffers are mapped using AADL event ports connected between AADL threads. This ports must not be connected outside the process.

Semaphores

Semaphores are mapped using a shared AADL data component between several AADL `thread` components. The shared `data` component must use a concurrency protocol by defining the `Concurrency_Control_Protocol` property.

4.5.7 Protocols

You can describe which protocol you want to use in your system using a protocol layer. You specify the protocol layer using `virtual bus` components.

FIXME – complete once the work around virtual bus is finalized.

4.6 POK AADL library

POK provides an AADL library for rapid prototyping of partitioned embedded architectures. This library contains predefines components associated with relevant properties to generate a partitioned architecture.

The file that contains this AADL library is located in `misc/aadl-library.aadl`.

4.7 Examples

Examples of AADL models can be found in the `examples` directory of the POK archive.

Chapter 5

Configuration directives

This chapter details the different configuration directives for kernel and partitions. The configuration of kernel and partitions is made using C code. You must write it carefully since a mistake can have significant impacts in terms of safety or security.

Most of the time, the C configuration code will be macros in global variables. The purpose of this chapter is to detail each variable. If you use generated code, the configuration code is mostly generated in `deployment.c` and `deployment.h` files.

5.1 Automatic configuration from ARINC653 XML files

You can automatically generate the configuration of your kernel using ARINC653 XML deployment files. For that, we designed a tool that analyzes ARINC653 XML files and automatically produce the C configuration code (`deployment.h` and `deployment.c`).

However, the configuration produced is not as complete as the one generated from AADL models. Indeed, ARINC653 XML files do not contain enough information to generate the whole configuration and is not sufficient to generate the configuration of partitions. However, this is a good way to have basic configuration files that can be improved by manual editing.

The tool is located in the `misc` directory of POK releases. You can use it as it:

```
misc/arinc653-xml-conf.pl arinc653-configuration-file.xml
```

When it is invoked, this program automatically produces two files: `deployment.h` and `deployment.c`. These files must be compiled with the kernel for its automatic configuration.

5.2 Common configuration

The following macros can be defined for both partitions and kernel:

- `POK_GENERATED_CODE`: specify that the code compiled has been generated from AADL so that we can restrict and avoid the use of some functions. This maccro is automatically added by *Ocarina* when it generates code from AADL models.

5.3 Kernel configuration

5.3.1 Services activation

You can define which capabilities you want in the kernel by defining some macros. Depending on which maccro you define, it will add services and capabilities in your kernel. It was made to make a very tight kernel and ease verification/certification efforts.

When you use code generation capabilities, these declarations are automatically created in the `deployment.h` file.

- `POK_NEEDS_PARTITIONS` maccro indicates that you need partitioning services. It implies that you define configuration macros and variables for the partitioning service.
- `POK_NEEDS_SCHED` maccro specifies that you need the scheduler.
- `POK_NEEDS_PCI` maccro specifies that kernel will include services to use PCI devices.
- `POK_NEEDS_IO` maccro specifies that input/output service must be activated so that some partitions will be allowed to perform i/o.
- `POK_NEEDS_DEBUG` maccro specifies that debugging information are activated. Additional output will be produced.
- `POK_NEEDS_LOCKOBJECTS` maccro specifies that you need the lockobject service. It must be defined if you use mutexes or semaphores.
- `POK_NEEDS_THREADS` maccro that thread service must be activated.
- `POK_NEEDS_GETTICK` maccro that time service must be activated (interrupt frame on timer interrupt is installed and clock is available).
- `POK_NEEDS_SCHED_RR` : the Round Robin scheduling policy is included in the kernel.
- `POK_NEEDS_SCHED_RMS` : the Rate Monotonic Scheduling scheduling policy is included in the kernel.
- `POK_NEEDS_SCHED_EDF` : the Earliest Deadline First scheduling policy is included in the kernel.
- `POK_NEEDS_SCHED_LLF` : the Last Laxity First scheduling protocol is included in the kernel.
- `POK_NEEDS_SCHED_STATIC` : the static scheduling protocol is included in the kernel.
- `POK_NEEDS_PORTS_SAMPLING` : the sampling ports service for inter-partitions communication is included.

- `POK_NEEDS_PORTS_QUEUEING` : the queueing ports service for inter-partitions communication is included.

5.3.2 General configuration

Number of threads

The `POK_CONFIG_NB_THREADS` maccro specifies the number of threads in the system. This represents how many threads can be handled in the kernel.

The values must be computed like this : number of threads in your system + 2. In fact, in this maccro, you must add 2 additional threads : the kernel thread and the idle thread.

Number of lockobjects

The `POK_CONFIG_NB_LOCKOBJECTS` maccro specifies the number of lockobjects the kernel would manage. It is the sum of potential semaphores, mutexes or ARINC653 events.

5.3.3 Partitions configuration

5.3.4 Number of partitions

The `POK_CONFIG_NB_PARTITIONS` maccro specifies the number of partitions handled in the kernel.

Threads allocation across partitions

The `POK_CONFIG_PARTITIONS_NTHREADS` maccro specifies how many threads would resides in partitions. This declaration is an array, each value of the array corresponds to the number of threads inside a partition.

An example is given below. In this example, we consider that we have 4 partitions. The first, second and third partitions handle two threads while the last partition has 4 threads.

Number of nodes

The `POK_CONFIG_NB_NODES` specifies the number of nodes in the distributed system if you use a such architecture. It is useful if you have more than one node and use network capabilities.

```
#define POK_CONFIG_PARTITIONS_NTHREADS {2,2,2,4}
```

Lockobjects allocation across partitions

The `POK_CONFIG_PARTITIONS_NLOCKOBJECTS` specifies the number of lock objects for each partition. This declaration is an array, each value *n* specifies how many lock objects we have for partition *n*.

There is an example of the use of this configuration directive. Here, the first partition will have one lockobject while the second partition will have three lockobjects.

```
#define POK_CONFIG_PARTITIONS_NLOCKOBJECTS {1,3}
```

Scheduler of each partition (level 1 of scheduling)

The `POK_CONFIG_PARTITIONS_SCHEDULER` specifies the scheduler used in each partition. This declaration is an array, each value *n* corresponds to the scheduler used for partition *n*.

There is an example below. Here, the four partitions used the *Round-Robin* scheduler.

```
#define POK_CONFIG_PARTITIONS_SCHEDULER {POK_SCHED_RR,POK_SCHED_RR,POK_SCHED_RR,POK_SCHED_RR}
```

Scheduler of partitions (level 0 of scheduling)

The scheduling of partitions is specified with several macros.

The `POK_CONFIG_SCHEDULING_NBSLOTS` specifies the number of time frames allocated for partitions execution.

The `POK_CONFIG_SCHEDULING_SLOTS` specifies the size (in milliseconds) of each slot.

The `POK_CONFIG_SCHEDULING_SLOTS_ALLOCATION` specified the allocation of each slot. In other words, which partition is scheduling at which slot. The declaration is an array and the value *n* specifies which partition uses the slot *n*.

The `POK_CONFIG_MAJOR_FRAME` specifies the major frame, the time when inter-partitions ports are flushed. It corresponds to the end of a scheduling cycle.

An example is provided below. Here, we have four partitions. We declare 4 slots of 500ms. The first slot is for the first partition, the second slot for the second partition and so on. The major frame (time when scheduling slots are repeated) is 2s (2000ms).

```
#define POK_CONFIG_SCHEDULING_SLOTS {500,500,500,500}

#define POK_CONFIG_SCHEDULING_SLOTS_ALLOCATION {0,1,2,3}

#define POK_CONFIG_SCHEDULING_NBSLOTS 4

#define POK_CONFIG_SCHEDULING_MAJOR_FRAME 2000
```

Partitions size

The `POK_CONFIG_PARTITIONS_SIZE` maccro specifies an array with partitions size in bytes. The declaration is an array, each value *n* represent the size of partition *n*.

There is an example of a such declaration below. Here, we have 4 partitions. The three first partition have a size of 80000 bytes while the last one has a size of 85000 bytes.

```
#define POK_CONFIG_PARTITIONS_SIZE {80000,80000,80000,85000}
```

5.3.5 Inter-partitions ports communication

For inter-partitions communication, we introduce several concepts:

- The **node identifier** is a unique number for each node.
- The **global port identifier** is a unique number for each port in the whole distributed system. This unique identifier identifies each port of each node.
- The **local port identifier** is a unique number for each port on the local node **only**. It identifies each inter-partition communication port on the local kernel.

So, for each node, you must specify in the kernel:

- The node identifier of the current node
- The number of nodes in the distributed system
- The number of inter-partitions ports in the distributed system
- The number of inter-partitions ports on the local node
- All identifiers of global ports
- All identifiers of local ports
- The association between global ports and nodes
- The association between global ports and local ports
- The association between local ports and global ports

Current node identifier

The identifier of the current node is specified with the `POK_CONFIG_LOCAL_NODE` maccro.

When you use code generation, this declaration is automatically created in the `deployment.h` file.

Number of global ports

The number of global ports in the distributed system is specified with the `POK_CONFIG_NB_GLOBAL_PORTS`. It indicates the number of global ports in the system.

When you use code generation, this declaration is automatically created in the `deployment.h` file.

Number of local ports

The number of local ports in the current node is specified using the `POK_CONFIG_NB_PORTS` maccro. It specifies the number of ports on the local node.

When you use code generation, this declaration is automatically created in the `deployment.h` file.

Local ports identifiers

The local ports identifiers are specified in an enum with the identifier `pok_port_local_identifier_t`. In this enum, you must ALWAYS add an identifier for an invalid identifier called `invalid_identifier`. Note that this enum declaration specifies the local ports of the current node and consequently, it is dependent on each node communication requirements.

When you use code generation, this declaration is automatically created in the `deployment.h` file.

There is an example of a such enum declaration:

```
typedef enum
{
  node1_partition_secret_outgoing = 0,
  node1_partition_topsecret_outgoing = 1,
  node1_partition_unclassified_outgoing = 2,
  invalid_local_port = 3
} pok_port_local_identifier_t;
```

Global ports identifiers

The global ports identifiers is specified using an enum called `pok_port_identifier_t`. This enum declaration must be **THE SAME** on all node of the distributed system.

When you use code generation, this declaration is automatically created in the `deployment.h` file.

There is an example of a such enum declaration:

```
typedef enum
{
  node1_partition_secret_outgoing_global = 0,
  node1_partition_topsecret_outgoing_global = 1,
  node1_partition_unclassified_outgoing_global = 2,
  node2_partition_secret_incoming_global = 3,
  node2_partition_topsecret_incoming_global = 4,
  node2_partition_unclassified_incoming_global = 5
} pok_port_identifier_t;
```

Node identifiers

The node identifiers are specified by declaring the `pok_node_identifier_t` type. It contains the value of each node identifier. Please also note that the `POK_CONFIG_LOCAL_NODE` value must be in this enum declaration. This enum declaration is **THE SAME** on all nodes of the distributed system.

When you use code generation, this declaration is automatically created in the `deployment.h` file.

There is an example of a such declaration

```
typedef enum
{
  node1 = 0,
  node2 = 1
} pok_node_identifier_t;
```

Associate local ports with global ports

We specify the global port of each local port with the `pok_ports_identifiers` array. An example is given below:

```
uint8_t pok_ports_identifiers[POK_CONFIG_NB_PORTS] =
         {node1_partition_secret_outgoing,
         node1_partition_topsecret_outgoing,
         node1_partition_unclassified_outgoing};
```

Here, the first local port of the current node corresponds to the `node1_partition_secret_outgoing` global port.

When you use code generation, this declaration is automatically created in the `deployment.c` file.

Specify local ports routing (local ports to global ports)

For each local port, we specify the number of destinations. Since there can be more than one recipient to a sending port, we specify how many ports should receive data. We specify that with the `pok_ports_nb_destinations` array.

Then, we specify the local port routing policy with the `pok_ports_destinations` array. In this array, each value is a pointer to another array that contains the recipient global port values.

An example is given below. Here, the first local port has one recipient. The recipient list is specified with the first elements of the `pok_ports_destinations` array, which is the `node1_partition_secret_outgoing_deployment_destinations` array. Thus, we can see that the recipient port identifier is `node2_partition_secret_incoming_global`.

```
uint8_t node1_partition_secret_outgoing_deployment_destinations[1] =
   {node2_partition_secret_incoming_global};
uint8_t node1_partition_secret_partport[1] =
   {node1_partition_secret_outgoing};
uint8_t node1_partition_topsecret_outgoing_deployment_destinations[1] =
   {node2_partition_topsecret_incoming_global};
uint8_t node1_partition_unclassified_outgoing_deployment_destinations[1] =
   {node2_partition_unclassified_incoming_global};

uint8_t pok_ports_nb_destinations[POK_CONFIG_NB_PORTS] = {1,1,1};

uint8_t* pok_ports_destinations[POK_CONFIG_NB_PORTS] =
   {node1_partition_secret_outgoing_deployment_destinations,
    node1_partition_topsecret_outgoing_deployment_destinations,
    node1_partition_unclassified_outgoing_deployment_destinations};
```

Convert local port to global ports

The association (conversion) between each local and global ports is given with the `pok_local_ports_to_global_ports` array. For each local port identifier, we specify the associated global port value.

An example is given below. Here, the first local port corresponds to the global port identifier `node1_partition_secret_outgoing_global`.

```
uint8_t pok_local_ports_to_global_ports[POK_CONFIG_NB_PORTS] =
   {node1_partition_secret_outgoing_global,
    node1_partition_topsecret_outgoing_global,
    node1_partition_unclassified_outgoing_global};
```

When you use code generation, this declaration is automatically created in the `deployment.c` file.

Convert global port to local port

It is sometimes needed to convert a global port value to a local port. You can have this information with the `pok_global_ports_to_local_ports`.

The definition of this array is different on all nodes. It specifies the local port identifier on the current node with each global port. If the global port is not on the current node, we specify the `invalid_port` value.

An example is given below. We can see that the three last ports are not located on the current node.

```
uint8_t pok_global_ports_to_local_ports[POK_CONFIG_NB_GLOBAL_PORTS] =
{node1_partition_secret_outgoing,
 node1_partition_topsecret_outgoing,
 node1_partition_unclassified_outgoing,
 invalid_local_port,
 invalid_local_port,
 invalid_local_port};
```

When you use code generation, this declaration is automatically created in the `deployment.c` file.

Location of each global port

The location of each global port is specified with the `pok_ports_nodes` array. It indicates, for each port, the associated node identifier.

In the following example, it shows that the three first global ports are located on the node 0 and the other on the node 1.

```
uint8_t pok_ports_nodes[POK_CONFIG_NB_GLOBAL_PORTS] =
   {0,0,0,1,1,1};
```

When you use code generation, this declaration is automatically created in the `deployment.c` file.

Specify the port type

The kernel must know the kind of each port (queuing or sampling). We specify that requirement with the `pok_ports_kind` array. There is an example of a such declaration below.

```
pok_port_kind_t pok_ports_kind[POK_CONFIG_NB_PORTS] =
{POK_PORT_KIND_SAMPLING,POK_PORT_KIND_SAMPLING,POK_PORT_KIND_SAMPLING};
```

Here, the three local ports are sampling ports. You can have three kind of ports:

1. **Sampling ports** (`POK_PORT_KIND_SAMPLING`) : stores data but does not queue them.
2. **Queuing ports** (`POK_PORT_KIND_QUEUEING`) : queues every new instance of the data.
3. **Virtual ports** (`POK_PORT_KIND_VIRTUAL`) : this port is not stored in the kernel and this is a virtual port. This port belongs to another machine. We add it only to create the routing policy in the distributed network. You cannot write or read data on/from virtual ports, only get the port identifier associated with them.

When you use code generation, this declaration is automatically created in the `deployment.c` file.

Specify ports names

When the developer calls ports instanciation, he can specify a port name. For that reason, the kernel must know the name associated with each port.

This information is provided by the `pok_ports_names` declaration. It contains the name of each local port.

There is an example of a such declaration.

```
char* pok_ports_names[POK_CONFIG_NB_PORTS] =
   {"node1_partition_secret_outgoing",
    "node1_partition_topsecret_outgoing",
    "node1_partition_unclassified_outgoing"};
```

When you use code generation, this declaration is automatically created in the `deployment.c` file.

Specify port usage for each partition

The inter-partition ports are dedicated to some partitions. Consequently, we have to specify in the configuration code which partition is allowed to read/write which port.

We do that with two arrays : `pok_ports_nb_ports_by_partition` and `pok_ports_by_partition`.

The `pok_ports_nb_ports_by_partition` indicates for each partition, the number of ports allocated. In the same manner, the `pok_ports_by_partition` indicate an array that contains the global ports identifiers allowed for this partition.

An example is provided. In this example, we see that the first partition has one port and the identifier of this port is `node1_partition_secret_outgoing`.

```
uint8_t node1_partition_secret_partport[1] =
   {node1_partition_secret_outgoing};

uint8_t node1_partition_topsecret_partport[1] =
   {node1_partition_topsecret_outgoing};

uint8_t node1_partition_unclassified_partport[1] =
   {node1_partition_unclassified_outgoing};


uint8_t pok_ports_nb_ports_by_partition[POK_CONFIG_NB_PARTITIONS] =
   {1,1,1};

uint8_t* pok_ports_by_partition[POK_CONFIG_NB_PARTITIONS] =
   {node1_partition_secret_partport,
    node1_partition_topsecret_partport,
    node1_partition_unclassified_partport};
```

When you use code generation, this declaration is automatically created in the `deployment.c` file.

5.4 Libpok (partition runtime)

5.5 Configuration

You define the configuration policy by defining some C-style macros. There are the list of useful macros:

- `POK_CONFIG_NB_THREADS`: specify the number of threads contained in the partition.
- `POK_CONFIG_NB_BUFFERS`: Specify the number of buffers used in the libpok (intra-partition communication).
- `POK_CONFIG_NB_SEMAPHORES`: Specify the number of semaphores used in the libpok (intra-partition communication).
- `POK_CONFIG_NB_BLACKBOARDS`: Specify the number of blackboard we use for intra-partition communications.
- `POK_CONFIG_NB_EVENTS`: Specify the number of events we use for intra-partition communications.
- `POK_CONFIG_ALLOCATOR_NB_SPACES`: Indicate the number of spaces we should reserve in the memory allocator. Since the memory allocator tries to reach determinism, the number of space is fixed. So, you have to specify how many spaces you want by defining this maccro.
- `POK_CONFIG_ALLOCATOR_MEMORY_SIZE`: Indicate which amount of memory must be reserved for the memory allocator.
- `POK_HW_ADDR`: Define the hardware address of the ethernet card. This maccro is useful if the partition implements a device driver for a network device. In POK and its libpok layer, we use it for the RTL8029 device driver.

5.6 Services activation

To activate *libpok* services, you must define some macros. By default, you don't have any services. You activate service by defining macros. Thus, it ensures that each partition contains only required services and avoid any memory overhead in partitions.

These macros have the form `POK_NEEDS_...`. There is a list of these macros:

- `POK_NEEDS_RTL8029`: activate the functions of the device driver that support the *Realtek* 8029 ethernet card.
- `POK_NEEDS_STDLIB`: activate services of the standard library (everything you can find in `libpok/include/libc/stdlib.h`).
- `POK_NEEDS_STDIO`: activate the services of the standard Input/Output library (printf, etc.). You can find available functions in `libpok/include/libc/stdio.h`).

- `POK_NEEDS_IO`: needs functions to perform I/O. These functions are just system calls and ask the kernel to perform them. The partition **CANNOT** make any I/O by itself.
- `POK_NEEDS_TIME`: activate functions that handle time.
- `POK_NEEDS_THREADS`: activate functions relative to threads.
- `POK_NEEDS_PORTS_VIRTUAL`: activate functions for virtual ports management. Virtual ports are handled by the kernel. So, activated functions in the libpok are just system call to the kernel to get the port routing policy. Since virtual ports represent ports that are located on other nodes, this maccro should be used only by partitions that actually implement network device drivers.
- `POK_NEEDS_PORTS_SAMPLING`: activate interfacing functions with the kernel to use sampling ports.
- `POK_NEEDS_PORTS_QUEUEING`: activate interfacing functions with the kernel to use queueing ports.
- `POK_NEEDS_ALLOCATOR` : activate the memory allocator of the partition. This service can be configured with `POK_CONFIG_ALLOCATOR...` macros.
- `POK_NEEDS_ARINC653_PROCESS`: activate the process service of the ARINC653 layer.
- `POK_NEEDS_ARINC653_BLACKBOARD`: activate the blackboard service of the ARINC653 layer
- `POK_NEEDS_ARINC653_BUFFER`: activate the buffer service of the ARINC653 layer.
- `POK_NEEDS_ARINC653_SEMAPHORE`: activate the semaphore service of the ARINC653 layer.
- `POK_NEEDS_ARINC653_QUEUEING`: activate the queueing service of the ARINC653 layer.
- `POK_NEEDS_ARINC653_SAMPLING`: activate the sampling ports service of the ARINC653 layer.
- `POK_NEEDS_ARINC653_ERROR`: activate the error service of the ARINC653 layer (health monitoring functions)
- `POK_NEEDS_BLACKBOARDS`: activate the blackboard service of POK (intra-partition communication)
- `POK_NEEDS_SEMAPHORES`: activate the semaphore service of POK (intra-partition communication)
- `POK_NEEDS_BUFFERS`: activate the buffer service of POK (intra-partition communication)

- `POK_NEEDS_ERROR_HANDLING`: activate the error handling service in POK.
- `POK_NEEDS_DEBUG`: activate debug mode.
- `POK_NEEDS_LIBMATH`: activate the libmath, functions that are available in regular service by passing the `-lm` flag to the compiler. See `libpok/include/libm.h` file for the list of functions.

Chapter 6

Examples

6.1 Assurance Quality

At each source code change, the developper must compile and check that examples compile fine on all supported architectures.

Consequently, the available examples with the release compiles. Sometimes, you can experience some errors since the examples are not run. If you think you find a bug, please report it to the developper team.

6.2 List of provided examples

This section details each example and the services they use. These examples are available in each release of POK.

- **arinc653-blackboard** : test the blackboard service of the ARINC653 layer. This test relies on an AADL model that describe a blackboards between two tasks.
- **arinc653-buffer**: test the buffer service of ARINC653. It uses AADL models.
- **arinc653-queueing**: test ARINC653 queuing ports with AADL.
- **arinc653-sampling**: test ARINC653 sampling ports with AADL.
- **arinc653-threads**: test ARINC653 processes instanciation. Uses AADL models.
- **case-study-aerotech09**: An ARINC653 examples case study for the AEROTECH09 conference. It uses two partitions that communication temperature across inter-partitions communications.
- **case-study-sigada09**: a system that contain three partitions with different design patterns (non-preemptive scheduler, ravenscar partition, queued buffers). This example was used as use-case for a publication in the SIGAda conference (SIGAda09).

- **case-study-mils**: a distributed with two nodes that communicate data at different security levels. Data are encrypted using cipher algorithms provided by the libpok layer.
- **case-study-ardupilot**: a case-study made from application code found in the ardupilot project. See. `http://code.google.com/p/ardupilot/` for documentation about this application code.
- **case-study-integrated**: a case-study that shows we can use POK for real avionics architecture. For that, we define an avionics architecture using AADL and generates code for POK using Ocarina. The initial model is defined by the Software Engineering Instute (SEI). See `http://www.aadl.info/aadl/currentsite/examplemodel.html#Famil` for more information about this initial model. Note that we convert this model from AADLv1 to AADLv2 to make it working with Ocarina/POK.
- **data-arrays** Test the use of array types and their use in communication with queuing ports. Use AADL models to describe types.
- **data-arrays2** Test the use of array types and their use with sampling ports. Use AADL model to describe types.
- **esterel** Use of a third-party language as application-layer. In this case, we use Esterel generated code. Use AADL models.
- **events** Test the use of events ports between threads located in the same partition. DO NOT use AADL models.
- **exceptions-handled**: Test the exceptions catching (recovery handlers). Use AADL models.
- **heterogeneous-partitions**: Define two partitions with different architectures. Demonstrate that the build system can generate, build and run different modules that have different architectures.
- **libmath**: Test the inclusion of the libmath library in the libpok layer. Use AADL model.
- **lustre-academic**: Test the inclusion of Lustre application code inside partition. Use AADL models.
- **middleware-blackboard**: Test the use of blackboard service. Use AADL models.
- **middleware-buffer**: Test the use of buffer service. Use AADL models.
- **middleware-buffer-timed**: Test the use of buffer service with timeout. Use AADL models.
- **middleware-queueing**: Test the use of queuing port service. Use AADL models.

- **middleware-queueing-timed**: Test the use of queuing port service with timeout. Use AADL models.
- **middleware-sampling**: Test the use of sampling port service. Use AADL models.
- **mutexes**: Test mutex service (POK layer). Use AADL models.
- **mutexes-timed**: Test mutex service with timeout. Use AADL models.
- **network**: Test network driver (rtl8029) on x86 architecture. Use AADL models.
- **partitions-scheduling**: Example with different schedulers. Use AADL models.
- **partitions-threads**: Test thread instanciation (POK layer). Use AADL models.
- **semaphores**: Test the use of semaphors (POK layer). Do not use AADL models.
- **simulink**: Test the inclusion of simulink code. Do not work since it needs a dedicated runtime to work. It needs additional services in the libpok layer to work. This additional work is not so difficult to provide, we just need time !

Chapter 7

Architecture

7.1 Directories hierarchy

The project is organized with a hierarchy of several directories:

- **examples**: sample code that uses pok and libpok. Code of examples is mostly generated from AADL models by Ocarina.
- **kernel**: code of the kernel that provides time and space partitioning services.
- **libpok**: code of libpok, the runtime of each partition. It contains libc, POSIX and arinc653 compliant abstraction layers.
- **misc**: misc files, such as makefiles, various tools to compile POK and so on.

7.2 ”Schyzophrenic” architecture

POK can be used as an executive (i.e a kernel that contains different tasks but does not provide partitioning functionnalities) or a partitioned architecture (a kernel isolates tasks in so-called partitions in terms of space and time).

Moreover, it was designed to support several API and services. But you can finely tune the kernel to avoid unused services, reduce memory footprint and ease certification/verification efforts.

Next sections discusses the different architectures that can be used.

7.2.1 Partitioned architecture

The partitioned architecture pattern can be used with POK. In that case, the kernel will execute several partitions on top of the POK kernel and provide time and space partitioning across partitions.

Each partition contains their memory allocators, their runtime and ressources (the so-called *libpok* part). Partitions can have different scheduling algorithms to schedule their tasks.

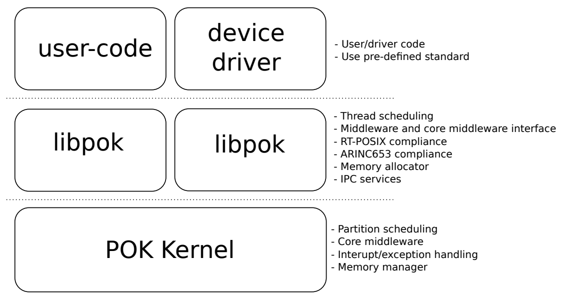

Figure 7.1: The different pok layers

In that case, the kernel provides communication isolation across partitions as well as space isolation (each partition has its own memory segment).

The overall architecture is illustrated in figure 7.1. The kernel executes the partitions, each partition contains its application code. Drivers are executed in partitions and don't reside inside the kernel.

To build a such architecture, you must have:

- For each partition
  - The application code
  - The configuration code
- For the kernel
  - The configuration code

Then, each part of the system is compiled and integrated, as depicted in figure 7.2. The kernel is compiled and each partitions is compiled. Eeach part produces a binary file. Since POK relies on the ELF file format, each binary of each part is compiled into an ELF file.

Then, we integrate **ALL** ELF files to produce a single bootable binary so that the final binary contains different binaries: the code for the kernel and the code of all partitions. Since POK relies on the ELF file format, the final ELF file contains other ELF files. The organization of the final binary is depiceted in figure 7.3.

When kernel boots, it loads each elf file of each partition in a different memory segment to achieve space isolation. So, each ELF file of each partition is loaded in a single and protected memory area of the system.

7.2.2 Executive architecture

At this time, the execute architecture pattern is not finished.

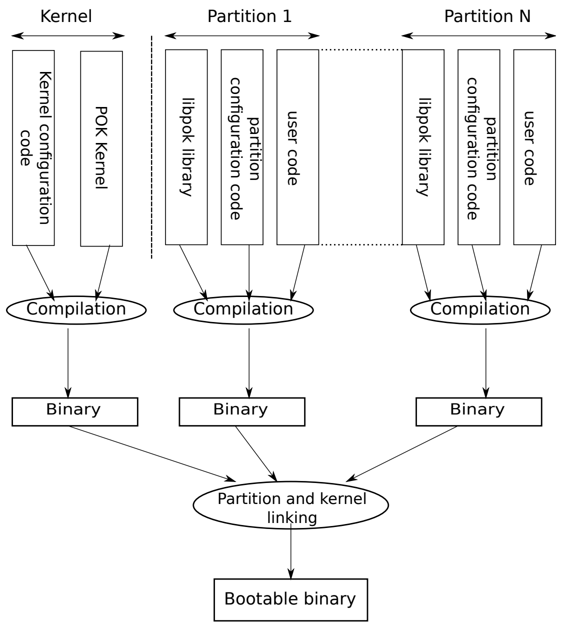

Figure 7.2: Build steps for a partitioned system

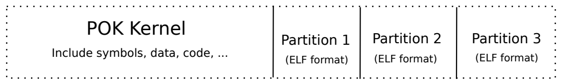

Figure 7.3: ELF file format of a POK system

7.3 Kernel services

7.3.1 Partitioning service

The partitioning service of POK isolates code in time and space. Each partition has one or more time slots to execute their code and they are isolated in a memory segment.

Using this design guideline, one partition cannot access the memory of other partitions (and *vice-versa*). During partitions initialization, POK automatically creates a memory segment for each partition and copy its code into this protected space.

However, partitions can communicate with other partitions using so-called ports. Inter-partitions ports are also supervised by the kernel in order to avoid unallowed communication channel. See section 7.3.4 for more information.

Partitions have time slots to execute their threads. During this execution time, they schedule their threads according to their own scheduling protocol so that partitions can schedule their threads in an independent way. This scheduling strategy is often described as a hierarchical scheduling.

7.3.2 Thread service

The thread service executes tasks. The system is built to execute a predefined number of tasks. When using partitioning services, each partitions has a predefined amount of tasks.

The scheduler can be preemptive so that tasks can interrupt each other. The thread service can start, stop or pause a task (sleep).

7.3.3 Time service

The time service provides an efficient way to manage the time on your machine. It is used by the scheduler to scheduler partitions and tasks according to their timing requirements (period, execution time and so on).

7.3.4 Communication service

The kernel provides communication services. It allows partitions and threads to communicate. The communication service is achieved using *ports*. *Out* ports (ports that send data) can have several destinations while *in* ports (ports that receive data) can have only one source.

Data are sent and received on this ports. The kernel configuration specifies the owner of a port, its destination and its size.

If you use partitioning service, each port is dedicated to a partition. Consequently, when creating the port, the kernel checks that requested port belongs to the partition.

Communication using network

When using the network, the port must be bound to a network interface so that data from/to the port will be sent over the network. The binding between a port and a network interface is specified in the kernel configuration.

Please note that in POK, when you are using partitioning services, device drivers are executed in partitions.

7.3.5 Scheduling service

The scheduling service schedules tasks and partitions according to their timing requirements. It relies on the time service.

Partitions are scheduled using a cyclic scheduling algorithm.

Partitions threads are scheduled using a Round-Robin, RMS or other available scheduling algorithms.

7.4 libpok services

7.4.1 Thread management

Thread management consist in interfacing functions with the kernel. It provides functions to start/suspend/stop a thread. It provides also locking services for mutexes/semaphores and so on.

7.4.2 Communication service

Libpok provides two kind of communication services:

- **Inter-partition communication** which consists in kernel-interfacing functions to use kernel communication ports.
- **Intra-partition communication service** which provides communication facilities to communicate inside a partition.

In the following, we detail intra-partition communication services.

Intra-partition communication service provides four communication patterns:

1. **Buffer** : thread send data. New data are queued according to a specific queueing policy. Items are dequeued when a task reads the buffer. We can store several instance of the same data.

   You need to define the `POK_NEEDS_BUFFERS` maccro to activate this service.

2. **Blackboard** : a shared memory space to store a data. New instances of the data replace the older value. We can store only one instance of the same data.

   You need to define the `POK_NEEDS_BLACKBOARDS` maccro to activate this service.

3. **Events** : are used to synchronized tasks. It corresponds to POSIX mutexes and conditions.

   You need to define the `POK_NEEDS_EVENTS` maccro to activate this service.

4. **Semaphores** : counting semaphores, as in the POSIX standard.

   You need to define the `POK_NEEDS_SEMAPHORES` maccro to activate this service.

7.4.3 Memory allocator

POK also provides a memory allocator. This memory allocator was designed to be deterministic and highly configurable. You define the amount of memory for the memory allocator and the number of memory slices that can be allocated.

Consequently, the memory allocator can be configured with different maccros. The service is activated by defining the `POK_CONFIG_NEEDS_ALLOCATOR` maccro. Then, the `POK_CONFIG_ALLOCATOR_MEMORY_SIZE` is used to specify the amount of memory dedicated for the memory allocator. Finally the `POK_CONFIG_ALLOCATOR_NB_SPACES` specifies the number of spaces you can allocate with the memory allocator.

This memory allocator can be used with the legacy layer (with the `pok_allocator_allocate()` or `pok_allocator_free()` functions) or with the C-library layer (`malloc()`, `free()`, `calloc()`).

7.4.4 Mathematic library service

We also add mathematic functions to ease the portability of third-party code. These functions were imported from the NetBSD[1] project. It provides all necessary functions to perform math operations (`sqrt()`, ...).

To enable the libmath functions, you must define the maccro `POK_NEEDS_LIBMATH`.

To have the complete list, please refer to the libpok reference manual available on each POK release. A list of these functions is also available in this document, in chapter 8.

7.4.5 Protocols

The libpok layer contains predefined protocols to marshall/unmarshall application data before sending them on the network. These protocols library could be used for several purposes: encrypt data before sending it on an unsecure network, adapt application data to constrained protocols such as CORBA, ...

These protocols can be automatically used through AADL models and appropriate properties associated to AADL data ports on AADL process components. To have more information about AADL and protocol binding, see section 4.

At this time, the libpok layer is focuses on crypto and provides the following protocols:

- Ceasar
- DES
- SSL

For each protocol, we have:

- A function to marshall data.
- A function to unmarshall data.

[1] http://www.netbsd.org

- An associated type if the protocol needs a special data type to store marshalled values.

Marshalling functions and types are described in their header files (see `des.h`, `ssl.h`, `ceasar.h` and so on). If there is no associated marshalling type, then, the marshall/unmarshall functions uses the same type as the application type or not particular type are required.

Details of each protocol can be found in the API section (chapter 8).

Chapter 8

POK API

8.1 Core C

8.1.1 Error values

```c

extern uint32_t errno;

#ifndef __POK_ERRNO_H__
#define __POK_ERRNO_H__

typedef enum {
  POK_ERRNO_OK = 0,
  POK_ERRNO_EINVAL = 1,

  POK_ERRNO_UNAVAILABLE = 2,
  POK_ERRNO_PARAM = 3,
  POK_ERRNO_TOOMANY = 5,
  POK_ERRNO_EPERM = 6,
  POK_ERRNO_EXISTS = 7,

  POK_ERRNO_ERANGE = 8,
  POK_ERRNO_EDOM = 9,
  POK_ERRNO_HUGE_VAL = 10,

  POK_ERRNO_EFAULT = 11,

  POK_ERRNO_THREAD = 49,
  POK_ERRNO_THREADATTR = 50,

  POK_ERRNO_TIME = 100,

  POK_ERRNO_PARTITION_ATTR = 200,

  POK_ERRNO_PORT = 301,
  POK_ERRNO_NOTFOUND = 302,
  POK_ERRNO_DIRECTION = 303,
  POK_ERRNO_SIZE = 304,
  POK_ERRNO_DISCIPLINE = 305,
  POK_ERRNO_PORTPART = 307,
  POK_ERRNO_EMPTY = 308,
  POK_ERRNO_KIND = 309,
  POK_ERRNO_FULL = 311,
  POK_ERRNO_READY = 310,
  POK_ERRNO_TIMEOUT = 250,
  POK_ERRNO_MODE = 251,

  POK_ERRNO_LOCKOBJ_UNAVAILABLE = 500,
  POK_ERRNO_LOCKOBJ_NOTREADY = 501,
  POK_ERRNO_LOCKOBJ_KIND = 502,
  POK_ERRNO_LOCKOBJ_POLICY = 503,

  POK_ERRNO_PARTITION_MODE = 601,

  POK_ERRNO_PARTITION = 401
} pok_ret_t;

#endif
```

8.1.2 Memory Allocation

```
#include <types.h>

#ifdef POK_NEEDS_ALLOCATOR

/*
 * This file contains memory allocation functionnalities.
 * You can tweak/tune the memory allocator with the following macros:
 *   - POK_CONFIG_ALLOCATOR_NB_SPACES : the number of memory spaces
 *     that can be allocated. It can corresponds to the successive
 *     call of malloc() or calloc() or pok_allocator_allocate()
 *   - POK_CONFIG_ALLOCATOR_MEMORY_SIZE : the amount of memory
 *     the allocator can allocate
 */

void *pok_allocator_allocate(size_t needed_size);
/*
 * This function allocates memory. The argument is the amount
 * of memory the user needs. This function is called by libc
 * functions malloc() and calloc()
 */

void pok_allocator_free(void *ptr);
/*
 * This function frees memory. The argument is a previously
 * allocated memory chunk. Be careful, the time required
 * to free the memory is indeterministic, you should not
 * free memory if your program has strong timing requirements.
 */

#endif
```

8.1.3 Threads

```
#define __POK_THREAD_H__

#include <core/dependencies.h>

#include <core/syscall.h>
#include <errno.h>
#include <types.h>

#define POK_THREAD_DEFAULT_PRIORITY 42

#define POK_DEFAULT_STACK_SIZE 2048

typedef struct {
  uint8_t priority; /* Priority is from 0 to 255 */
  uint8_t processor_affinity;
  void *entry; /* entrypoint of the thread  */
  uint64_t period;
  uint64_t deadline;
  uint64_t time_capacity;
  uint32_t stack_size;
  uint32_t state;
} pok_thread_attr_t;

void pok_thread_init(void);
pok_ret_t pok_thread_create(uint32_t *thread_id, const pok_thread_attr_t *attr);
pok_ret_t pok_thread_sleep(const pok_time_t us);
pok_ret_t pok_thread_sleep_until(const pok_time_t us);
pok_ret_t pok_thread_lock();
pok_ret_t pok_thread_unlock(const uint32_t thread_id);
unsigned int pok_thread_current(void);
void pok_thread_start(void (*entry)(), uint32_t id);
void pok_thread_switch(uint32_t elected_id);
pok_ret_t pok_thread_wait_infinite();
void pok_thread_wrapper();
pok_ret_t pok_thread_attr_init(pok_thread_attr_t *attr);
pok_ret_t pok_thread_period();
pok_ret_t pok_thread_id(uint32_t *thread_id);
void pok_thread_init(void);
pok_ret_t pok_thread_status(const uint32_t thread_id, pok_thread_attr_t *attr);
pok_ret_t pok_thread_delayed_start(const uint32_t thread_id,
                                   const pok_time_t us);
pok_ret_t pok_thread_set_priority(const uint32_t thread_id,
                                  const uint32_t priority);
pok_ret_t pok_thread_resume(const uint32_t thread_id);

#define pok_thread_sleep_until(time)
\
  pok_syscall2(POK_SYSCALL_THREAD_SLEEP_UNTIL, (uint32_t)time, 0)

#define pok_thread_wait_infinite() pok_thread_suspend()

#define pok_thread_suspend()
\
  pok_syscall2(POK_SYSCALL_THREAD_SUSPEND, NULL, NULL)

```

```
#define pok_thread_suspend_target(thread_id)
\
  pok_syscall2(POK_SYSCALL_THREAD_SUSPEND_TARGET, thread_id, 0)

/*
 * Similar to: pok_ret_t      pok_thread_suspend (void);
 */

#define pok_thread_restart(thread_id)
\
  pok_syscall2(POK_SYSCALL_THREAD_RESTART, thread_id, 0)
/*
 * similar to:
 * pok_ret_t      pok_thread_restart (uint32_t thread_id);
 */

#define pok_thread_stop_self() pok_syscall2(POK_SYSCALL_THREAD_STOPSELF, 0, 0)
/*
 * similar to:
 * pok_ret_t      pok_thread_stop_self ();
 */

#define pok_thread_stop(id) pok_syscall2(POK_SYSCALL_THREAD_STOP, id, NULL)
/*
 * similar to: pok_ret_t      pok_thread_stop (const uint32_t tid);
 */

#endif /* __POK_THREAD_H__ */
```

8.1.4 Error handling

```

#include <errno.h>
#include <types.h>

#define POK_ERROR_MAX_LOGGED 100

typedef struct {
  uint8_t error_kind;
  uint32_t failed_thread;
  uint32_t failed_addr;
  char *msg;
  uint32_t msg_size;
} pok_error_status_t;

typedef struct {
  uint32_t thread;
  uint32_t error;
  pok_time_t when;
} pok_error_report_t;

extern pok_error_report_t pok_error_reported[POK_ERROR_MAX_LOGGED];

#define POK_ERROR_KIND_DEADLINE_MISSED 10
#define POK_ERROR_KIND_APPLICATION_ERROR 11
#define POK_ERROR_KIND_NUMERIC_ERROR 12
```

```
#define POK_ERROR_KIND_ILLEGAL_REQUEST 13
#define POK_ERROR_KIND_STACK_OVERFLOW 14
#define POK_ERROR_KIND_MEMORY_VIOLATION 15
#define POK_ERROR_KIND_HARDWARE_FAULT 16
#define POK_ERROR_KIND_POWER_FAIL 17
#define POK_ERROR_KIND_PARTITION_CONFIGURATION 30
#define POK_ERROR_KIND_PARTITION_INIT 31
#define POK_ERROR_KIND_PARTITION_SCHEDULING 32
#define POK_ERROR_KIND_PARTITION_PROCESS 33
#define POK_ERROR_KIND_KERNEL_INIT 50
#define POK_ERROR_KIND_KERNEL_SCHEDULING 51

pok_ret_t pok_error_handler_create();
void pok_error_ignore(const uint32_t error_id, const uint32_t thread_id);
void pok_error_confirm(const uint32_t error_id, const uint32_t thread_id);

void pok_error_log(const uint32_t error_id, const uint32_t thread_id);

void pok_error_raise_application_error(char *msg, uint32_t msg_size);

/**
 * pok_error_get returns POK_ERRNO_OK if the error pointer
 * was registered and an error was registered.
 * It also returns POK_ERRNO_UNAVAILABLE if the pointer
 * was not registered or if nothing was detected
 */
pok_ret_t pok_error_get(pok_error_status_t *status);
```

8.1.5 Inter-partitions communication

```

#ifndef __POK_LIBPOK_PORTS_H__
#define __POK_LIBPOK_PORTS_H__

#include <core/lockobj.h>
#include <core/syscall.h>
#include <errno.h>
#include <types.h>

typedef enum {
  POK_PORT_DIRECTION_IN = 1,
  POK_PORT_DIRECTION_OUT = 2
} pok_port_directions_t;

typedef pok_queueing_discipline_t pok_port_queueing_discipline_t;

typedef enum {
  POK_PORT_KIND_QUEUEING = 1,
  POK_PORT_KIND_SAMPLING = 2,
  POK_PORT_KIND_VIRTUAL = 2,
  POK_PORT_KIND_INVALID = 10
} pok_port_kinds_t;

#ifdef POK_NEEDS_PORTS_VIRTUAL
pok_ret_t pok_port_virtual_create(char *name, pok_port_id_t *id);

pok_ret_t pok_port_virtual_destination(const pok_port_id_t id, const uint32_t n,
                                       uint32_t *result);

pok_ret_t pok_port_virtual_nb_destinations(const pok_port_id_t id,
                                           uint32_t *result);

pok_ret_t pok_port_virtual_get_global(const pok_port_id_t local,
                                      pok_port_id_t *global);

pok_ret_t pok_port_virtual_node(uint32_t port, uint8_t *node);

#endif

#ifdef POK_NEEDS_MAC_ADDR
pok_ret_t pok_node_mac_addr(uint8_t *node_id, char *addr);
#endif

#ifdef POK_NEEDS_PORTS_QUEUEING
/* Queueing port functions */
typedef struct {
  pok_port_size_t size;
  pok_port_direction_t direction;
  uint8_t nb_messages;
  uint8_t waiting_processes;
} pok_port_queueing_status_t;

pok_ret_t
pok_port_queueing_create(char *name, const pok_port_size_t size,
                         const pok_port_direction_t direction,
                         const pok_port_queueing_discipline_t discipline,
                         pok_port_id_t *id);

pok_ret_t pok_port_queueing_receive(const pok_port_id_t id,
                                    const uint64_t timeout,
                                    const pok_port_size_t maxlen, void *data,
                                    pok_port_size_t *len);

pok_ret_t pok_port_queueing_send(const pok_port_id_t id, const void *data,
                                 const pok_port_size_t len,
                                 const uint64_t timeout);

#define pok_port_queueing_status(id, status)
\
  pok_syscall2(POK_SYSCALL_MIDDLEWARE_QUEUEING_STATUS, (uint32_t)id,
\
               (uint32_t)status)
/*
 * Similar to:
 * pok_ret_t pok_port_queueing_status (const pok_port_id_t id, const
 * pok_port_queueing_status_t*      status);
 */

#define pok_port_queueing_id(name, id)
\
  pok_syscall2(POK_SYSCALL_MIDDLEWARE_QUEUEING_ID, (uint32_t)name, (uint32_t)id)
/*
 * Similar to:
```

```
 * pok_ret_t pok_port_queueing_id      (char* name, pok_port_id_t* id);
 */
#endif

#ifdef POK_NEEDS_PORTS_SAMPLING
/* Sampling port functions */

typedef struct {
  pok_port_size_t size;
  pok_port_direction_t direction;
  uint64_t refresh;
  bool_t validity;
} pok_port_sampling_status_t;

pok_ret_t pok_port_sampling_create(char *name, const pok_port_size_t size,
                                   const pok_port_direction_t direction,
                                   const uint64_t refresh, pok_port_id_t *id);

pok_ret_t pok_port_sampling_write(const pok_port_id_t id, const void *data,
                                  const pok_port_size_t len);

pok_ret_t pok_port_sampling_read(const pok_port_id_t id, void *message,
                                 pok_port_size_t *len, bool_t *valid);

#define pok_port_sampling_id(name, id)
\
  pok_syscall2(POK_SYSCALL_MIDDLEWARE_SAMPLING_ID, (uint32_t)name, (uint32_t)id)
/*
 *    Similar to
 *  pok_ret_t pok_port_sampling_id      (char* name, pok_port_id_t* id);
 */

#define pok_port_sampling_status(id, status)
\
  pok_syscall2(POK_SYSCALL_MIDDLEWARE_SAMPLING_STATUS, (uint32_t)id,
\
               (uint32_t)status)
/*
 * Similar to:
 * pok_ret_t pok_port_sampling_status (const pok_port_id_t id, const
 * pok_port_sampling_status_t*                status);
 */
#endif

#endif
```

8.1.6 Intra-partitions communications

Configuration

Blackboards

```
#define __POK_USER_BLACKBOARD_H__

#ifdef POK_NEEDS_MIDDLEWARE
#ifdef POK_NEEDS_BLACKBOARDS
```

```c

#include <errno.h>
#include <types.h>

typedef struct {
  pok_size_t size;
  pok_bool_t empty;
  pok_range_t waiting_processes;
  pok_size_t index;
  pok_bool_t ready;
  pok_event_id_t lock;
} pok_blackboard_t;

typedef struct {
  pok_port_size_t msg_size;
  pok_bool_t empty;
  pok_range_t waiting_processes;
} pok_blackboard_status_t;

pok_ret_t pok_blackboard_create(char *name, const pok_size_t msg_size,
                                pok_blackboard_id_t *id);

pok_ret_t pok_blackboard_read(const pok_blackboard_id_t id,
                              const uint64_t timeout, void *data,
                              pok_port_size_t *len);

pok_ret_t pok_blackboard_display(const pok_blackboard_id_t id,
                                 const void *message,
                                 const pok_port_size_t len);

pok_ret_t pok_blackboard_clear(const pok_blackboard_id_t id);

pok_ret_t pok_blackboard_id(char *name, pok_blackboard_id_t *id);

pok_ret_t pok_blackboard_status(const pok_blackboard_id_t id,
                                pok_blackboard_status_t *status);
#endif
#endif
#endif
```

Buffers

```c
#define __POK_USER_BUFFER_H__

#ifdef POK_NEEDS_MIDDLEWARE
#ifdef POK_NEEDS_BUFFERS

#define POK_BUFFER_DISCIPLINE_FIFO 1
#define POK_BUFFER_DISCIPLINE_PRIORITY 2

#include <errno.h>
#include <types.h>

#include <core/lockobj.h>

typedef struct {
  pok_bool_t ready;
  pok_bool_t empty;
  pok_bool_t full;
  pok_size_t size;
  pok_size_t index;
  pok_port_size_t off_b;
  pok_port_size_t off_e;
  pok_port_size_t msgsize;
  pok_range_t waiting_processes;
  pok_queueing_discipline_t discipline;
  pok_event_id_t lock;
} pok_buffer_t;

typedef struct {
  pok_range_t nb_messages;
  pok_range_t max_messages;
  pok_size_t message_size;
  pok_range_t waiting_processes;
} pok_buffer_status_t;

pok_ret_t pok_buffer_create(char *name, const pok_port_size_t size,
                            const pok_port_size_t msg_size,
                            const pok_queueing_discipline_t discipline,
                            pok_buffer_id_t *id);

pok_ret_t pok_buffer_receive(const pok_buffer_id_t id, const uint64_t timeout,
                             void *data, pok_port_size_t *len);

pok_ret_t pok_buffer_send(const pok_buffer_id_t id, const void *data,
                          const pok_port_size_t len, const uint64_t timeout);

pok_ret_t pok_port_buffer_status(const pok_buffer_id_t id,
                                 const pok_buffer_status_t *status);

pok_ret_t pok_buffer_id(char *name, pok_buffer_id_t *id);

#endif
#endif

#endif
```

Events

```
#define __POK_LIBPOK_EVENT_H__

#include <core/dependencies.h>

#include <errno.h>
#include <types.h>

pok_ret_t pok_event_create(pok_event_id_t *id);
pok_ret_t pok_event_wait(pok_event_id_t id, uint64_t timeout_us);
pok_ret_t pok_event_broadcast(pok_event_id_t id);
pok_ret_t pok_event_signal(pok_event_id_t id);
pok_ret_t pok_event_lock(pok_event_id_t id);
pok_ret_t pok_event_unlock(pok_event_id_t id);
```

```

#endif
```

Semaphores

```
#define __POK_KERNEL_SEMAPHORE_H__

#include <core/dependencies.h>

#ifdef POK_NEEDS_SEMAPHORES

#include <core/lockobj.h>
#include <errno.h>
#include <types.h>

pok_ret_t pok_sem_create(pok_sem_id_t *id, const pok_sem_value_t current_value,
                         const pok_sem_value_t max_value,
                         const pok_queueing_discipline_t discipline);

pok_ret_t pok_sem_wait(pok_sem_id_t id, uint64_t timeout_us);

pok_ret_t pok_sem_signal(pok_sem_id_t id);

pok_ret_t pok_sem_id(char *name, pok_sem_id_t *id);

pok_ret_t pok_sem_status(pok_sem_id_t id, pok_sem_status_t *status);

#endif

#endif
```

8.1.7 C-library

Standard Input/Output

```
#define __POK_LIBC_STDIO_H__

#include "stdarg.h"

#define EOF -1

int vprintf(const char *format, va_list args);
;

int printf(const char *format, ...) __attribute__((format(printf, 1, 2)));

int getChar();

#endif /* __POK_LIBC_STDIO_H_ */
```

Standard Lib

```c
#define __POK_STDLIB_H__

#include <types.h>

#define RAND_MAX 256

int rand();
void *calloc(size_t count, size_t size);
void *malloc(size_t size);
void free(void *ptr);

#endif
```

String functions

```c
#define __POK_LIBC_STRING_H__

#include <types.h>

char *itoa(int value, char *buff, int radix);
int atoi(const char *str);
void *memcpy(void *dest, const void *src, size_t count);
void *memset(void *dest, unsigned char val, size_t count);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t size);
size_t strlen(const char *s);
char *strcpy(char *dest, const char *str);
char *strncpy(char *dest, const char *str, size_t size);
int memcmp(const void *v1, const void *v2, size_t n);

/*
** XXX: TO REMOVE
*/
int streq(char *s1, char *s2);

#endif
```

8.1.8 Math functions

```c

#ifndef __POK_LIBM_H__
#define __POK_LIBM_H__

#include <types.h>

struct exception {
  int type;
  char *name;
  double arg1;
  double arg2;
  double retval;
};

```

```
#define FP_NAN 1
#define FP_INFINITE 2
#define FP_NORMAL 3
#define FP_SUBNORMAL 4
#define FP_ZERO 5

#define DOMAIN 1
#define SING 2
#define OVERFLOW 3
#define UNDERFLOW 4
#define TLOSS 5
#define PLOSS 6

#define fpclassify(x)
\
  (sizeof(x) == sizeof(float) ? __fpclassifyf((float)(x))
\
                              : __fpclassifyd((double)(x)))

extern int __fpclassifyf(float);
extern int __fpclassifyd(double);
extern int __fpclassify(long double);

double acos(double x);
float acosf(float x);
double acosh(double x);
float acoshf(float x);
double asin(double x);
float asinf(float x);
double asinh(double x);
float asinhf(float x);
double atan(double x);
float atanf(float x);
double atan2(double y, double x);
float atan2f(float y, float x);
double atanh(double x);
float atanhf(float x);
double cbrt(double x);
float cbrtf(float x);
double ceil(double x);
float ceilf(float x);
double copysign(double x, double y);
float copysignf(float x, float y);
double cos(double x);
float cosf(float x);
double cosh(double x);
float coshf(float x);
double drem(double x, double y);
float dremf(float x, float y);
double erf(double x);
float erff(float x);
double exp(double x);
float expf(float x);
double expm1(double x);
float expm1f(float x);
double fabs(double x);
float fabsf(float x);
```

```
int finite(double x);
int finitef(float x);
double floor(double x);
float floorf(float x);
double frexp(double x, int *eptr);
float frexpf(float x, int *eptr);
double gamma(double x);
float gammaf(float x);
double gamma_r(double x, int *signgamp);
float gammaf_r(float x, int *signgamp);
double hypot(double x, double y);
float hypotf(float x, float y);
int ilogb(double x);
int ilogbf(float x);
int isinf(double x);
int isinff(float x);
int isnan(double x);
int isnanf(float x);
double j0(double x);
float j0f(float x);
double j1(double x);
float j1f(float x);
double jn(int n, double x);
float jnf(int n, float x);
double ldexp(double value, int exp0);
float ldexpf(float value, int exp0);
double lgamma(double x);
float lgammaf(float x);
double lgamma_r(double x, int *signgamp);
float lgammaf_r(float x, int *signgamp);
double log(double x);
float logf(float x);
double log10(double x);
float log10f(float x);
double log2(double x);
float log2f(float x);
double logb(double x);
float logbf(float x);
double log1p(double x);
float log1pf(float x);
double ldexp(double value, int exp0);
float ldexpf(float value, int exp0);
int matherr(struct exception *x);
float modff(float x, float *iptr);
double modf(double x, double *iptr);
double nextafter(double x, double y);
float nextafterf(float x, float y);
double pow(double x, double y);
float powf(float x, float y);
double remainder(double x, double y);
float remainderf(float x, float y);
#ifdef _SCALB_INT
double scalb(double x, int fn);
#else
double scalb(double x, double fn);
#endif
#ifdef _SCALB_INT
```

```c
float scalbf(float x, int fn);
#else
float scalbf(float x, float fn);
#endif
double rint(double x);
float rintf(float x);
double round(double x);
float roundf(float x);
double scalbn(double x, int n);
float scalbnf(float x, int n);
double significand(double x);
float significandf(float x);
double sin(double x);
float sinf(float x);
double sinh(double x);
float sinhf(float x);
double sqrt(double x);
float sqrtf(float x);
double tan(double x);
float tanf(float x);
double tanh(double x);
float tanhf(float x);
double trunc(double x);
float truncf(float x);

#endif

#endif /* POK_NEEDS_LIBMATH */
```

8.1.9 Protocol functions

```c
#define __LIBPOK_PROTOCOLS_H__

/**
 * \file    libpok/protocols/protocols.h
 * \author  Julien Delange
 * \date    2009
 * \brief   Protocols to marshall/unmarshall data
 *
 * This file is a general-purpose file to include all
 * protocols in the same time. Protocols functions
 * provides features to encode and decode messages
 * before sending data through partitions. It is
 * especially useful when you want to encrypt data
 * over the network before sending or adapt application
 * data to a particular protocol.
 *
 * For each protocol, we have:
 *   - One function to marshall data
 *   - One function to unmarshall data
 *   - One data type associated with the crypto protocol.
 *     This data type is used to store data when marshalling
 *     data and used as an input to unmarshall data.
 *
 * More documentation is available in the user manual.
 */
```

```

/*
 * The DES crypto protocol
 */
#include <protocols/des.h>

/*
 * The Blowfish crypto protocol
 */
#include <protocols/blowfish.h>

/*
 * The Ceasar crypto protocol
 */
#include <protocols/ceasar.h>

#endif
```

```
#define __LIBPOK_PROTOCOLS_CEASAR_H__

/**
 * \file    libpok/include/protocols/ceasar.h
 * \author  Julien Delange
 * \date    2009
 * \brief   Ceasar crypto protocol.
 *
 * This is a very basic crypto protocol that just
 * change the order of bytes in data. There is no
 * public/private key, the algorithm is known
 * by the attacker so that it's a very weak crypto
 * protocol.
 * Interested people can gather more information
 * about this protocol on:
 * http://en.wikipedia.org/wiki/Caesar_cipher
 *
 * We don't provide an associated marshalling type
 * for the Ceasar protocol since the crypted size
 * is the same than the uncrypted size.
 */

#include <types.h>

#ifdef POK_NEEDS_PROTOCOLS_CEASAR

/**
 * Function that uncrypts data
 */
void pok_protocols_ceasar_unmarshall(void *crypted_data,
                                     pok_size_t crypted_size,
                                     void *uncrypted_data,
                                     size_t *uncrypted_size);

/**
 * Function that encrypts data
 */
void pok_protocols_ceasar_marshall(void *uncrypted_data,
                                   pok_size_t uncrypted_size,
                                   void *crypted_data, size_t *crypted_size);

#endif

#endif
```

```
#define __LIBPOK_PROTOCOLS_DES_H__

/**
 * \file    libpok/protocols/des.h
 * \author  Julien Delange
 * \date    2009
 * \brief   DES protocol.
 *
 * Implementation of the very basic DES crypto
 * protocol. This is a symetric crypto protocol
 * with a shared key so that receiver and sender
 * share the same key.
 *
 * More information at:
 * http://en.wikipedia.org/wiki/Data_Encryption_Standard
 */

#include <types.h>

#define pok_protocols_des_data_t unsigned long long

#ifdef POK_NEEDS_PROTOCOLS_DES
/**
 * Function that uncrypts data.
 */
void pok_protocols_des_unmarshall(void *crypted_data, pok_size_t crypted_size,
                                  void *uncrypted_data, size_t *uncrypted_size);

/**
 * Function that crypts data.
 */
void pok_protocols_des_marshall(void *uncrypted_data, pok_size_t uncrypted_size,
                                void *crypted_data, size_t *crypted_size);

/**
 * The key for the DES protocol is on 8 bytes and is
 * defined by the macro POK_PROTOCOLS_DES_KEY
 */
#ifndef POK_PROTOCOLS_DES_KEY
#define POK_PROTOCOLS_DES_KEY
\
  { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef }
#endif

/**
 * The init vector for the DES protocol is on 8 bytes
 * defined by the macro POK_PROTOCOLS_DES_INIT
 */
#ifndef POK_PROTOCOLS_DES_INIT
#define POK_PROTOCOLS_DES_INIT
\
  { 0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10 }
#endif

#endif

#endif
```

```
#define __LIBPOK_PROTOCOLS_SSL_H__

#include <types.h>

/**
 * \file    libpok/protocols/ssl.h
 * \author  Julien Delange
 * \date    2009
 * \brief   SSL crypto protocol.
 *
 * More information at:
 * http://en.wikipedia.org/wiki/Transport_Layer_Security
 */

#ifdef POK_NEEDS_PROTOCOLS
void pok_protocols_ssl_unmarshall(void *crypted_data, pok_size_t crypted_size,
                                  void *uncrypted_data, size_t *uncrypted_size);

void pok_protocols_ssl_marshall(void *uncrypted_data, pok_size_t uncrypted_size,
                                void *crypted_data, size_t *crypted_size);

#define pok_protocols_ssl_data_t int

#endif

#endif
```

8.2 ARINC653 C

An ARINC653 layer is available for partitions. This section presents the C layer, an Ada layer is also available and described in section 8.3.

8.2.1 APEX types and constants

```
#define APEX_TYPES

#include <types.h>

#define SYSTEM_LIMIT_NUMBER_OF_PARTITIONS 32      /* module scope */
#define SYSTEM_LIMIT_NUMBER_OF_MESSAGES 512       /* module scope */
#define SYSTEM_LIMIT_MESSAGE_SIZE 8192            /* module scope */
#define SYSTEM_LIMIT_NUMBER_OF_PROCESSES 128      /* partition scope */
#define SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS 512 /* partition scope */
#define SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS 512  /* partition scope */
#define SYSTEM_LIMIT_NUMBER_OF_BUFFERS 256        /* partition scope */
#define SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS 256    /* partition scope */
```

```
#define SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES 256     /* partition scope */
#define SYSTEM_LIMIT_NUMBER_OF_EVENTS 256         /* partition scope */

/*----------------------*/
/* Base APEX types      */
/*----------------------*/
/*  The actual size of these base types is system specific and the
*/
/*  sizes must match the sizes used by the implementation of the
*/
/*  underlying Operating System.
*/
typedef unsigned char APEX_BYTE;      /* 8-bit unsigned */
typedef long APEX_INTEGER;            /* 32-bit signed */
typedef unsigned long APEX_UNSIGNED; /* 32-bit unsigned */
typedef long long APEX_LONG_INTEGER; /* 64-bit signed */
/*----------------------*/
/* General APEX types   */
/*----------------------*/
typedef enum {
  NO_ERROR = 0,       /*  request valid and operation performed     */
  NO_ACTION = 1,      /*  status of system unaffected by request    */
  NOT_AVAILABLE = 2,  /*  resource required by request unavailable  */
  INVALID_PARAM = 3,  /*  invalid parameter specified in request    */
  INVALID_CONFIG = 4, /*  parameter incompatible with configuration */
  INVALID_MODE = 5,   /*  request incompatible with current mode    */
  TIMED_OUT = 6       /*  time-out tied up with request has expired */
} RETURN_CODE_TYPE;
#define MAX_NAME_LENGTH 30
typedef char NAME_TYPE[MAX_NAME_LENGTH];
typedef void(*SYSTEM_ADDRESS_TYPE);
typedef APEX_BYTE *MESSAGE_ADDR_TYPE;
typedef APEX_INTEGER MESSAGE_SIZE_TYPE;
typedef APEX_INTEGER MESSAGE_RANGE_TYPE;
typedef enum { SOURCE = 0, DESTINATION = 1 } PORT_DIRECTION_TYPE;
typedef enum { FIFO = 0, PRIORITY = 1 } QUEUING_DISCIPLINE_TYPE;
typedef APEX_LONG_INTEGER
    SYSTEM_TIME_TYPE; /* 64-bit signed integer with a 1 nanosecond LSB */
#define INFINITE_TIME_VALUE -1

#endif
```

8.2.2 Partition management

```

#include <arinc653/process.h>
#include <arinc653/types.h>

#ifndef APEX_PARTITION
#define APEX_PARTITION
#define MAX_NUMBER_OF_PARTITIONS SYSTEM_LIMIT_NUMBER_OF_PARTITIONS
typedef enum {
  IDLE = 0,
  COLD_START = 1,
  WARM_START = 2,
  NORMAL = 3
} OPERATING_MODE_TYPE;

typedef APEX_INTEGER PARTITION_ID_TYPE;
typedef enum {
  NORMAL_START = 0,
  PARTITION_RESTART = 1,
  HM_MODULE_RESTART = 2,
  HM_PARTITION_RESTART = 3
} START_CONDITION_TYPE;

typedef struct {
  SYSTEM_TIME_TYPE PERIOD;
  SYSTEM_TIME_TYPE DURATION;
  PARTITION_ID_TYPE IDENTIFIER;
  LOCK_LEVEL_TYPE LOCK_LEVEL;
  OPERATING_MODE_TYPE OPERATING_MODE;
  START_CONDITION_TYPE START_CONDITION;
} PARTITION_STATUS_TYPE;

extern void GET_PARTITION_STATUS(
    /*out*/ PARTITION_STATUS_TYPE *PARTITION_STATUS,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);
extern void SET_PARTITION_MODE(
    /*in */ OPERATING_MODE_TYPE OPERATING_MODE,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);
#endif

#endif
```

8.2.3 Time management

```
#ifndef APEX_TIME
#define APEX_TIME

#include <arinc653/types.h>

/*----------------------------------------------------------------------*/
/* */
/* time constant definitions */
/* */
/*----------------------------------------------------------------------*/
/* implementation dependent */
/* these values are given as example */
/*----------------------------------------------------------------------*/
/* */
/* time type definitions */
/* */
/*----------------------------------------------------------------------*/
/*----------------------------------------------------------------------*/
/* */
/* time management services */
/* */
/*----------------------------------------------------------------------*/
/*----------------------------------------------------------------------*/
extern void TIMED_WAIT(
    /*in */ SYSTEM_TIME_TYPE delay_time,
    /*out*/ RETURN_CODE_TYPE *return_code);
/*----------------------------------------------------------------------*/
extern void PERIODIC_WAIT(
    /*out*/ RETURN_CODE_TYPE *return_code);
/*----------------------------------------------------------------------*/
extern void GET_TIME(
    /*out*/ SYSTEM_TIME_TYPE *system_time,
    /*out*/ RETURN_CODE_TYPE *return_code);
/*----------------------------------------------------------------------*/
void REPLENISH(SYSTEM_TIME_TYPE budget_time, RETURN_CODE_TYPE *return_code);
/*----------------------------------------------------------------------*/
#endif
#endif
```

8.2.4 Error handling

```
#ifndef APEX_ERROR
#define APEX_ERROR

#ifndef POK_NEEDS_ARINC653_PROCESS
#define POK_NEEDS_ARINC653_PROCESS 1
#endif

#include <arinc653/process.h>

#include <arinc653/types.h>

#define MAX_ERROR_MESSAGE_SIZE 64

typedef APEX_INTEGER ERROR_MESSAGE_SIZE_TYPE;

typedef APEX_BYTE ERROR_MESSAGE_TYPE[MAX_ERROR_MESSAGE_SIZE];

enum ERROR_CODE_VALUE_TYPE {
  DEADLINE_MISSED = 0,
  APPLICATION_ERROR = 1,
  NUMERIC_ERROR = 2,
  ILLEGAL_REQUEST = 3,
  STACK_OVERFLOW = 4,
  MEMORY_VIOLATION = 5,
  HARDWARE_FAULT = 6,
  POWER_FAIL = 7
};

typedef enum ERROR_CODE_VALUE_TYPE ERROR_CODE_TYPE;
/*------------------------------*/
/* error status type */
/*------------------------------*/
typedef struct {
  ERROR_CODE_TYPE ERROR_CODE;
  MESSAGE_SIZE_TYPE LENGTH;
  PROCESS_ID_TYPE FAILED_PROCESS_ID;
  SYSTEM_ADDRESS_TYPE FAILED_ADDRESS;
  ERROR_MESSAGE_TYPE MESSAGE;
} ERROR_STATUS_TYPE;

```

```
/*---------------------------------------------------------------*/
/* */
/* ERROR MANAGEMENT SERVICES */
/* */
/*---------------------------------------------------------------*/
/*---------------------------------------------------------------*/

extern void REPORT_APPLICATION_MESSAGE (MESSAGE_ADDR_TYPE MESSAGE,
                                        MESSAGE_SIZE_TYPE LENGTH,
                                        RETURN_CODE_TYPE *RETURN_CODE);

extern void CREATE_ERROR_HANDLER (SYSTEM_ADDRESS_TYPE ENTRY_POINT,
                                  STACK_SIZE_TYPE STACK_SIZE,
                                  RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_ERROR_STATUS (ERROR_STATUS_TYPE *ERROR_STATUS,
                              RETURN_CODE_TYPE *RETURN_CODE);

extern void RAISE_APPLICATION_ERROR (ERROR_CODE_TYPE ERROR_CODE,
                                     MESSAGE_ADDR_TYPE MESSAGE,
                                     ERROR_MESSAGE_SIZE_TYPE LENGTH,
                                     RETURN_CODE_TYPE *RETURN_CODE);
#endif
#endif
```

8.2.5 Process management

```

#include <arinc653/types.h>

#ifndef APEX_PROCESS
#define APEX_PROCESS

#define MAX_NUMBER_OF_PROCESSES SYSTEM_LIMIT_NUMBER_OF_PROCESSES
#define MIN_PRIORITY_VALUE 1
#define MAX_PRIORITY_VALUE 63
#define MAX_LOCK_LEVEL 16

typedef NAME_TYPE PROCESS_NAME_TYPE;

typedef APEX_INTEGER PROCESS_ID_TYPE;

typedef APEX_INTEGER LOCK_LEVEL_TYPE;

typedef APEX_UNSIGNED STACK_SIZE_TYPE;

typedef APEX_INTEGER WAITING_RANGE_TYPE;

typedef APEX_INTEGER PRIORITY_TYPE;

typedef enum {
  DORMANT = 0,
  READY = 1,
  RUNNING = 2,
  WAITING = 3
} PROCESS_STATE_TYPE;
```

```c

typedef enum { SOFT = 0, HARD = 1 } DEADLINE_TYPE;

typedef struct {
  SYSTEM_TIME_TYPE PERIOD;
  SYSTEM_TIME_TYPE TIME_CAPACITY;
  SYSTEM_ADDRESS_TYPE ENTRY_POINT;
  STACK_SIZE_TYPE STACK_SIZE;
  PRIORITY_TYPE BASE_PRIORITY;
  DEADLINE_TYPE DEADLINE;
  PROCESS_NAME_TYPE NAME;
} PROCESS_ATTRIBUTE_TYPE;

typedef struct {
  SYSTEM_TIME_TYPE DEADLINE_TIME;
  PRIORITY_TYPE CURRENT_PRIORITY;
  PROCESS_STATE_TYPE PROCESS_STATE;
  PROCESS_ATTRIBUTE_TYPE ATTRIBUTES;
} PROCESS_STATUS_TYPE;

extern void CREATE_PROCESS (
    /*in */ PROCESS_ATTRIBUTE_TYPE *ATTRIBUTES,
    /*out*/ PROCESS_ID_TYPE *PROCESS_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void SET_PRIORITY (
    /*in */ PROCESS_ID_TYPE PROCESS_ID,
    /*in */ PRIORITY_TYPE PRIORITY,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void SUSPEND_SELF (
    /*in */ SYSTEM_TIME_TYPE TIME_OUT,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void SUSPEND (
    /*in */ PROCESS_ID_TYPE PROCESS_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void RESUME (
    /*in */ PROCESS_ID_TYPE PROCESS_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void STOP_SELF ();

extern void STOP (
    /*in */ PROCESS_ID_TYPE PROCESS_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void START (
    /*in */ PROCESS_ID_TYPE PROCESS_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void DELAYED_START (
    /*in */ PROCESS_ID_TYPE PROCESS_ID,
    /*in */ SYSTEM_TIME_TYPE DELAY_TIME,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

```

```
extern void LOCK_PREEMPTION(
     /*out*/ LOCK_LEVEL_TYPE *LOCK_LEVEL,
     /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void UNLOCK_PREEMPTION(
     /*out*/ LOCK_LEVEL_TYPE *LOCK_LEVEL,
     /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_MY_ID(
     /*out*/ PROCESS_ID_TYPE *PROCESS_ID,
     /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_PROCESS_ID(
     /*in */ PROCESS_NAME_TYPE PROCESS_NAME[MAX_NAME_LENGTH],
     /*out*/ PROCESS_ID_TYPE *PROCESS_ID,
     /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_PROCESS_STATUS(
     /*in */ PROCESS_ID_TYPE PROCESS_ID,
     /*out*/ PROCESS_STATUS_TYPE *PROCESS_STATUS,
     /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

#endif
#endif
```

8.2.6 Blackboard service (intra-partition communication)

```

/*-------------------------------------------------------------------*/
/*                                                                   */
/* BLACKBOARD constant and type definitions and management services  */
/*                                                                   */
/*-------------------------------------------------------------------*/

#ifndef APEX_BLACKBOARD
#define APEX_BLACKBOARD

#ifndef POK_NEEDS_ARINC653_PROCESS
#define POK_NEEDS_ARINC653_PROCESS
#endif

#include <arinc653/process.h>
#include <arinc653/types.h>

#define MAX_NUMBER_OF_BLACKBOARDS SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS

typedef NAME_TYPE BLACKBOARD_NAME_TYPE;

typedef APEX_INTEGER BLACKBOARD_ID_TYPE;

typedef enum { EMPTY = 0, OCCUPIED = 1 } EMPTY_INDICATOR_TYPE;

typedef struct {
  EMPTY_INDICATOR_TYPE EMPTY_INDICATOR;
  MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE;
  WAITING_RANGE_TYPE WAITING_PROCESSES;
} BLACKBOARD_STATUS_TYPE;

extern void CREATE_BLACKBOARD(
    /*in */ BLACKBOARD_NAME_TYPE BLACKBOARD_NAME,
    /*in */ MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE,
    /*out*/ BLACKBOARD_ID_TYPE *BLACKBOARD_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void DISPLAY_BLACKBOARD(
    /*in */ BLACKBOARD_ID_TYPE BLACKBOARD_ID,
    /*in */ MESSAGE_ADDR_TYPE MESSAGE_ADDR, /* by reference */
    /*in */ MESSAGE_SIZE_TYPE LENGTH,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void READ_BLACKBOARD(
    /*in */ BLACKBOARD_ID_TYPE BLACKBOARD_ID,
    /*in */ SYSTEM_TIME_TYPE TIME_OUT,
    /*out*/ MESSAGE_ADDR_TYPE MESSAGE_ADDR,
    /*out*/ MESSAGE_SIZE_TYPE *LENGTH,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void CLEAR_BLACKBOARD(
    /*in */ BLACKBOARD_ID_TYPE BLACKBOARD_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_BLACKBOARD_ID(
    /*in */ BLACKBOARD_NAME_TYPE BLACKBOARD_NAME,
    /*out*/ BLACKBOARD_ID_TYPE *BLACKBOARD_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_BLACKBOARD_STATUS(
    /*in */ BLACKBOARD_ID_TYPE BLACKBOARD_ID,
    /*out*/ BLACKBOARD_STATUS_TYPE *BLACKBOARD_STATUS,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

#endif

#endif
```

8.2.7 Buffer service (intra-partition communication)

```

/*----------------------------------------------------------------*/
/*                                                                */
/* BUFFER constant and type definitions and management services   */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_BUFFER
#define APEX_BUFFER

#ifndef POK_NEEDS_ARINC653_PROCESS
#define POK_NEEDS_ARINC653_PROCESS
#endif

#include <arinc653/process.h>
```

```c
#include <arinc653/types.h>

#define MAX_NUMBER_OF_BUFFERS SYSTEM_LIMIT_NUMBER_OF_BUFFERS

typedef NAME_TYPE BUFFER_NAME_TYPE;

typedef APEX_INTEGER BUFFER_ID_TYPE;

typedef struct {
  MESSAGE_RANGE_TYPE NB_MESSAGE;
  MESSAGE_RANGE_TYPE MAX_NB_MESSAGE;
  MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE;
  WAITING_RANGE_TYPE WAITING_PROCESSES;
} BUFFER_STATUS_TYPE;

extern void CREATE_BUFFER(
    /*in */ BUFFER_NAME_TYPE BUFFER_NAME,
    /*in */ MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE,
    /*in */ MESSAGE_RANGE_TYPE MAX_NB_MESSAGE,
    /*in */ QUEUING_DISCIPLINE_TYPE QUEUING_DISCIPLINE,
    /*out*/ BUFFER_ID_TYPE *BUFFER_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void SEND_BUFFER(
    /*in */ BUFFER_ID_TYPE BUFFER_ID,
    /*in */ MESSAGE_ADDR_TYPE MESSAGE_ADDR, /* by reference */
    /*in */ MESSAGE_SIZE_TYPE LENGTH,
    /*in */ SYSTEM_TIME_TYPE TIME_OUT,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void RECEIVE_BUFFER(
    /*in */ BUFFER_ID_TYPE BUFFER_ID,
    /*in */ SYSTEM_TIME_TYPE TIME_OUT,
    /*out*/ MESSAGE_ADDR_TYPE MESSAGE_ADDR,
    /*out*/ MESSAGE_SIZE_TYPE *LENGTH,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_BUFFER_ID(
    /*in */ BUFFER_NAME_TYPE BUFFER_NAME,
    /*out*/ BUFFER_ID_TYPE *BUFFER_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_BUFFER_STATUS(
    /*in */ BUFFER_ID_TYPE BUFFER_ID,
    /*out*/ BUFFER_STATUS_TYPE *BUFFER_STATUS,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

#endif
#endif
```

8.2.8 Event service (intra-partition communication)

```c
#ifndef APEX_ERROR
#define APEX_ERROR

#ifndef POK_NEEDS_ARINC653_PROCESS
```

```
#define POK_NEEDS_ARINC653_PROCESS 1
#endif

#include <arinc653/process.h>

#include <arinc653/types.h>

#define MAX_ERROR_MESSAGE_SIZE 64

typedef APEX_INTEGER ERROR_MESSAGE_SIZE_TYPE;

typedef APEX_BYTE ERROR_MESSAGE_TYPE[MAX_ERROR_MESSAGE_SIZE];

enum ERROR_CODE_VALUE_TYPE {
  DEADLINE_MISSED = 0,
  APPLICATION_ERROR = 1,
  NUMERIC_ERROR = 2,
  ILLEGAL_REQUEST = 3,
  STACK_OVERFLOW = 4,
  MEMORY_VIOLATION = 5,
  HARDWARE_FAULT = 6,
  POWER_FAIL = 7
};

typedef enum ERROR_CODE_VALUE_TYPE ERROR_CODE_TYPE;
/*------------------------------*/
/* error status type */
/*------------------------------*/
typedef struct {
  ERROR_CODE_TYPE ERROR_CODE;
  MESSAGE_SIZE_TYPE LENGTH;
  PROCESS_ID_TYPE FAILED_PROCESS_ID;
  SYSTEM_ADDRESS_TYPE FAILED_ADDRESS;
  ERROR_MESSAGE_TYPE MESSAGE;
} ERROR_STATUS_TYPE;

/*---------------------------------------------------------------------*/
/* */
/* ERROR MANAGEMENT SERVICES */
/* */
/*---------------------------------------------------------------------*/
/*---------------------------------------------------------------------*/

extern void REPORT_APPLICATION_MESSAGE(MESSAGE_ADDR_TYPE MESSAGE,
                                       MESSAGE_SIZE_TYPE LENGTH,
                                       RETURN_CODE_TYPE *RETURN_CODE);

extern void CREATE_ERROR_HANDLER(SYSTEM_ADDRESS_TYPE ENTRY_POINT,
                                 STACK_SIZE_TYPE STACK_SIZE,
                                 RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_ERROR_STATUS(ERROR_STATUS_TYPE *ERROR_STATUS,
                             RETURN_CODE_TYPE *RETURN_CODE);

extern void RAISE_APPLICATION_ERROR(ERROR_CODE_TYPE ERROR_CODE,
                                    MESSAGE_ADDR_TYPE MESSAGE,
                                    ERROR_MESSAGE_SIZE_TYPE LENGTH,
                                        RETURN_CODE_TYPE *RETURN_CODE);
#endif
#endif
```

8.2.9 Queuing ports service (inter-partition communication)

```

#ifndef POK_NEEDS_ARINC653_PROCESS
#define POK_NEEDS_ARINC653_PROCESS
#endif

#include <arinc653/process.h>
#include <arinc653/types.h>

#ifndef APEX_QUEUING
#define APEX_QUEUING
#define MAX_NUMBER_OF_QUEUING_PORTS SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS
typedef NAME_TYPE QUEUING_PORT_NAME_TYPE;
typedef APEX_INTEGER QUEUING_PORT_ID_TYPE;

typedef struct {
  MESSAGE_RANGE_TYPE NB_MESSAGE;
  MESSAGE_RANGE_TYPE MAX_NB_MESSAGE;
  MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE;
  PORT_DIRECTION_TYPE PORT_DIRECTION;
  WAITING_RANGE_TYPE WAITING_PROCESSES;
} QUEUING_PORT_STATUS_TYPE;

extern void CREATE_QUEUING_PORT(
    /*in */ QUEUING_PORT_NAME_TYPE QUEUING_PORT_NAME,
    /*in */ MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE,
    /*in */ MESSAGE_RANGE_TYPE MAX_NB_MESSAGE,
    /*in */ PORT_DIRECTION_TYPE PORT_DIRECTION,
    /*in */ QUEUING_DISCIPLINE_TYPE QUEUING_DISCIPLINE,
    /*out*/ QUEUING_PORT_ID_TYPE *QUEUING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void SEND_QUEUING_MESSAGE(
    /*in */ QUEUING_PORT_ID_TYPE QUEUING_PORT_ID,
    /*in */ MESSAGE_ADDR_TYPE MESSAGE_ADDR, /* by reference */
    /*in */ MESSAGE_SIZE_TYPE LENGTH,
    /*in */ SYSTEM_TIME_TYPE TIME_OUT,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void RECEIVE_QUEUING_MESSAGE(
    /*in */ QUEUING_PORT_ID_TYPE QUEUING_PORT_ID,
    /*in */ SYSTEM_TIME_TYPE TIME_OUT,
    /*out*/ MESSAGE_ADDR_TYPE MESSAGE_ADDR,
    /*out*/ MESSAGE_SIZE_TYPE *LENGTH,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_QUEUING_PORT_ID(
    /*in */ QUEUING_PORT_NAME_TYPE QUEUING_PORT_NAME,
    /*out*/ QUEUING_PORT_ID_TYPE *QUEUING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

```

```
extern void GET_QUEUING_PORT_STATUS(
    /*in */ QUEUING_PORT_ID_TYPE QUEUING_PORT_ID,
    /*out*/ QUEUING_PORT_STATUS_TYPE *QUEUING_PORT_STATUS,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void CLEAR_QUEUING_PORT(
    /*in */ QUEUING_PORT_ID_TYPE QUEUING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

#endif

#endif
```

8.2.10 Sampling ports service (inter-partition communication)

```

#include <arinc653/types.h>

/*----------------------------------------------------------------------*/
/*                                                                      */
/* SAMPLING PORT constant and type definitions and management services*/
/*                                                                      */
/*----------------------------------------------------------------------*/

#ifndef APEX_SAMPLING
#define APEX_SAMPLING

#define MAX_NUMBER_OF_SAMPLING_PORTS SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS

typedef NAME_TYPE SAMPLING_PORT_NAME_TYPE;

typedef APEX_INTEGER SAMPLING_PORT_ID_TYPE;

typedef enum { INVALID = 0, VALID = 1 } VALIDITY_TYPE;

typedef struct {
  SYSTEM_TIME_TYPE REFRESH_PERIOD;
  MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE;
  PORT_DIRECTION_TYPE PORT_DIRECTION;
  VALIDITY_TYPE LAST_MSG_VALIDITY;
} SAMPLING_PORT_STATUS_TYPE;

extern void CREATE_SAMPLING_PORT(
    /*in */ SAMPLING_PORT_NAME_TYPE SAMPLING_PORT_NAME,
    /*in */ MESSAGE_SIZE_TYPE MAX_MESSAGE_SIZE,
    /*in */ PORT_DIRECTION_TYPE PORT_DIRECTION,
    /*in */ SYSTEM_TIME_TYPE REFRESH_PERIOD,
    /*out*/ SAMPLING_PORT_ID_TYPE *SAMPLING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void WRITE_SAMPLING_MESSAGE(
    /*in */ SAMPLING_PORT_ID_TYPE SAMPLING_PORT_ID,
    /*in */ MESSAGE_ADDR_TYPE MESSAGE_ADDR, /* by reference */
    /*in */ MESSAGE_SIZE_TYPE LENGTH,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

```

```
extern void READ_SAMPLING_MESSAGE(
    /*in */ SAMPLING_PORT_ID_TYPE SAMPLING_PORT_ID,
    /*out*/ MESSAGE_ADDR_TYPE MESSAGE_ADDR,
    /*out*/ MESSAGE_SIZE_TYPE *LENGTH,
    /*out*/ VALIDITY_TYPE *VALIDITY,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_SAMPLING_PORT_ID(
    /*in */ SAMPLING_PORT_NAME_TYPE SAMPLING_PORT_NAME,
    /*out*/ SAMPLING_PORT_ID_TYPE *SAMPLING_PORT_ID,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

extern void GET_SAMPLING_PORT_STATUS(
    /*in */ SAMPLING_PORT_ID_TYPE SAMPLING_PORT_ID,
    /*out*/ SAMPLING_PORT_STATUS_TYPE *SAMPLING_PORT_STATUS,
    /*out*/ RETURN_CODE_TYPE *RETURN_CODE);

#endif

#endif
```

8.3 ARINC653 Ada

Since partitions can also be written in Ada, an ARINC653 Ada layer - APEX - is available. It is just a binding to the C implementation which files can be found in `libpok/ada/arinc653`.

Although the binding is complete, *Health monitoring*, *Module schedules* and a few other functions are not yet available in the C API.

Simply use `with APEX.xxx` in your source to use the `xxx` ARINC module.

8.3.1 APEX types and constants

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- This is a compilable Ada 95 specification for the APEX interface,
-- derived from section 3 of ARINC 653.
-- The declarations of the services given below are taken from the
-- standard, as are the enumerated types and the names of the others types.
-- However, the definitions given for these others types, and the
-- names and values given below for constants, are all implementation
-- specific.
-- All types have defining representation pragmas or clauses to ensure
-- representation compatibility with the C and Ada 83 bindings.
-- ---------------------------------------------------------------------------
-- --
-- Root package providing constant and type definitions
--
-- --
-- ---------------------------------------------------------------------------
with System;
                    -- This is the Ada 95 predefined C interface package
with Interfaces.C;
package APEX is
   pragma Pure;
   -- ---------------------------
   -- Domain limits             --
   -- ---------------------------
   --  Domain dependent
   --  These values define the domain limits and are implementation-dependent.
   System_Limit_Number_Of_Partitions     : constant := 32;
   -- module scope
   System_Limit_Number_Of_Messages       : constant := 512;
   -- module scope
   System_Limit_Message_Size             : constant := 16#10_0000#;
   -- module scope
   System_Limit_Number_Of_Processes      : constant := 1024;
   -- partition scope
   System_Limit_Number_Of_Sampling_Ports : constant := 1024;
   -- partition scope
   System_Limit_Number_Of_Queuing_Ports  : constant := 1024;
   -- partition scope
   System_Limit_Number_Of_Buffers        : constant := 512;
   -- partition scope
   System_Limit_Number_Of_Blackboards    : constant := 512;
   -- partition scope
   System_Limit_Number_Of_Semaphores     : constant := 512;
   -- partition scope
   System_Limit_Number_Of_Events         : constant := 512;
   -- partition scope
   -- ---------------------------
   -- Base APEX types           --
   -- ---------------------------
   --  The actual sizes of these base types are system-specific and must
   --  match those of the underlying Operating System.
   type APEX_Byte is new Interfaces.C.unsigned_char;
   type APEX_Integer is new Interfaces.C.long;
   type APEX_Unsigned is new Interfaces.C.unsigned_long;
   type APEX_Long_Integer is new Interfaces.Integer_64;
   -- If Integer_64 is not provided in package Interfaces, any implementation-
   -- defined alternative 64-bit signed integer type may be used.
   -- ---------------------------
   -- General APEX types       --
   -- ---------------------------
   type Return_Code_Type is (
      No_Error,        -- request valid and operation performed
      No_Action,       -- status of system unaffected by request
      Not_Available,   -- resource required by request unavailable
      Invalid_Param,   -- invalid parameter specified in request
      Invalid_Config, -- parameter incompatible with configuration
      Invalid_Mode,     -- request incompatible with current mode
      Timed_Out);       -- time-out tied up with request has expired
   pragma Convention (C, Return_Code_Type);
   Max_Name_Length : constant := 30;
   subtype Name_Type is String (1 .. Max_Name_Length);
   subtype System_Address_Type is System.Address;
   subtype Message_Addr_Type is System.Address;
   subtype Message_Size_Type is APEX_Integer range
      1 .. System_Limit_Message_Size;
   subtype Message_Range_Type is APEX_Integer range
      0 .. System_Limit_Number_Of_Messages;
   type Port_Direction_Type is (Source, Destination);
   pragma Convention (C, Port_Direction_Type);
   type Queuing_Discipline_Type is (Fifo, Priority);
   pragma Convention (C, Queuing_Discipline_Type);
   subtype System_Time_Type is APEX_Long_Integer;
   -- 64-bit signed integer with 1 nanosecond LSB
   Infinite_Time_Value : constant System_Time_Type;
   Aperiodic           : constant System_Time_Type;
   Zero_Time_Value     : constant System_Time_Type;
private
   Infinite_Time_Value : constant System_Time_Type := -1;
   Aperiodic           : constant System_Time_Type := 0;
   Zero_Time_Value     : constant System_Time_Type := 0;
end APEX;
```

8.3.2 Blackboards

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- BLACKBOARD constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Blackboards is
   Max_Number_Of_Blackboards : constant := System_Limit_Number_Of_Blackboards;
   subtype Blackboard_Name_Type is Name_Type;
   type Blackboard_Id_Type is private;
   Null_Blackboard_Id : constant Blackboard_Id_Type;
   type Empty_Indicator_Type is (Empty, Occupied);
   type Blackboard_Status_Type is record
       Empty_Indicator   : Empty_Indicator_Type;
       Max_Message_Size : Message_Size_Type;
       Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;
   procedure Create_Blackboard
      (Blackboard_Name : in Blackboard_Name_Type;
       Max_Message_Size : in Message_Size_Type;
       Blackboard_Id     : out Blackboard_Id_Type;
       Return_Code       : out Return_Code_Type);
   procedure Display_Blackboard
      (Blackboard_Id : in Blackboard_Id_Type;
       Message_Addr : in Message_Addr_Type;
       Length         : in Message_Size_Type;
       Return_Code    : out Return_Code_Type);
   procedure Read_Blackboard
      (Blackboard_Id : in Blackboard_Id_Type;
       Time_Out       : in System_Time_Type;
       Message_Addr : in Message_Addr_Type;
   -- The message address is passed IN, although the respective message is
   -- passed OUT
       Length         : out Message_Size_Type;
       Return_Code    : out Return_Code_Type);
   procedure Clear_Blackboard
      (Blackboard_Id : in Blackboard_Id_Type;
       Return_Code    : out Return_Code_Type);
   procedure Get_Blackboard_Id
      (Blackboard_Name : in Blackboard_Name_Type;
       Blackboard_Id    : out Blackboard_Id_Type;
       Return_Code      : out Return_Code_Type);
   procedure Get_Blackboard_Status
      (Blackboard_Id      : in Blackboard_Id_Type;
       Blackboard_Status : out Blackboard_Status_Type;
       Return_Code        : out Return_Code_Type);
private
   type Blackboard_Id_Type is new APEX_Integer;
   Null_Blackboard_Id : constant Blackboard_Id_Type := 0;
   pragma Convention (C, Empty_Indicator_Type);
   pragma Convention (C, Blackboard_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Blackboard, "CREATE_BLACKBOARD");
   pragma Import (C, Display_Blackboard, "DISPLAY_BLACKBOARD");
   pragma Import (C, Read_Blackboard, "READ_BLACKBOARD");
   pragma Import (C, Clear_Blackboard, "CLEAR_BLACKBOARD");
   pragma Import (C, Get_Blackboard_Id, "GET_BLACKBOARD_ID");
   pragma Import (C, Get_Blackboard_Status, "GET_BLACKBOARD_STATUS");
   -- END OF POK BINDINGS
end APEX.Blackboards;
```

8.3.3 Buffers

```
--                                  POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
```

```
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                        Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- BUFFER constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Buffers is
   Max_Number_Of_Buffers : constant := System_Limit_Number_Of_Buffers;
   subtype Buffer_Name_Type is Name_Type;
   type Buffer_Id_Type is private;
   Null_Buffer_Id : constant Buffer_Id_Type;
   type Buffer_Status_Type is record
       Nb_Message         : Message_Range_Type;
       Max_Nb_Message     : Message_Range_Type;
       Max_Message_Size : Message_Size_Type;
       Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;
   procedure Create_Buffer
      (Buffer_Name         : in Buffer_Name_Type;
       Max_Message_Size    : in Message_Size_Type;
       Max_Nb_Message      : in Message_Range_Type;
       Queuing_Discipline :  in Queuing_Discipline_Type;
       Buffer_Id           : out Buffer_Id_Type;
       Return_Code         : out Return_Code_Type);
   procedure Send_Buffer
      (Buffer_Id    : in Buffer_Id_Type;
       Message_Addr : in Message_Addr_Type;
       Length       : in Message_Size_Type;
       Time_Out     : in System_Time_Type;
       Return_Code : out Return_Code_Type);
   procedure Receive_Buffer
      (Buffer_Id    : in Buffer_Id_Type;
       Time_Out     : in System_Time_Type;
       Message_Addr : in Message_Addr_Type;
   -- The message address is passed IN, although the respective message is
   -- passed OUT
       Length       : out Message_Size_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Buffer_Id
      (Buffer_Name : in Buffer_Name_Type;
       Buffer_Id   : out Buffer_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Buffer_Status
      (Buffer_Id      : in Buffer_Id_Type;
       Buffer_Status : out Buffer_Status_Type;
       Return_Code    : out Return_Code_Type);
private
   type Buffer_Id_Type is new APEX_Integer;
   Null_Buffer_Id : constant Buffer_Id_Type := 0;
   pragma Convention (C, Buffer_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Buffer, "CREATE_BUFFER");
   pragma Import (C, Send_Buffer, "SEND_BUFFER");
   pragma Import (C, Receive_Buffer, "RECEIVE_BUFFER");
   pragma Import (C, Get_Buffer_Id, "GET_BUFFER_ID");
   pragma Import (C, Get_Buffer_Status, "GET_BUFFER_STATUS");
   -- END OF POK BINDINGS
end APEX.Buffers;
```

8.3.4 Events

```
--                                POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                       Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- EVENT constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Events is
   Max_Number_Of_Events : constant := System_Limit_Number_Of_Events;
   subtype Event_Name_Type is Name_Type;
   type Event_Id_Type is private;
   Null_Event_Id : constant Event_Id_Type;
   type Event_State_Type is (Down, Up);
   type Event_Status_Type is record
       Event_State        : Event_State_Type;
       Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;
   procedure Create_Event
      (Event_Name : in Event_Name_Type;
       Event_Id    : out Event_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Set_Event
      (Event_Id    : in Event_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Reset_Event
      (Event_Id    : in Event_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Wait_Event
      (Event_Id    : in Event_Id_Type;
       Time_Out    : in System_Time_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Event_Id
      (Event_Name : in Event_Name_Type;
       Event_Id    : out Event_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Event_Status
```

```
      (Event_Id      : in Event_Id_Type;
       Event_Status  : out Event_Status_Type;
       Return_Code   : out Return_Code_Type);
private
   type Event_Id_Type is new APEX_Integer;
   Null_Event_Id : constant Event_Id_Type := 0;
   pragma Convention (C, Event_State_Type);
   pragma Convention (C, Event_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Event, "CREATE_EVENT");
   pragma Import (C, Set_Event, "SET_EVENT");
   pragma Import (C, Reset_Event, "RESET_EVENT");
   pragma Import (C, Wait_Event, "WAIT_EVENT");
   pragma Import (C, Get_Event_Id, "GET_EVENT_ID");
   pragma Import (C, Get_Event_Status, "GET_EVENT_STATUS");
    -- END OF POK BINDINGS
end APEX.Events;
```

8.3.5 Health monitoring

```
--                              POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- ERROR constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Health_Monitoring is
   Max_Error_Message_Size : constant := 64;
   subtype Error_Message_Size_Type is APEX_Integer range
      1 .. Max_Error_Message_Size;
   type Error_Message_Type is
      array (Error_Message_Size_Type) of APEX_Byte;
   type Error_Code_Type is (
       Deadline_Missed,
       Application_Error,
       Numeric_Error,
       Illegal_Request,
       Stack_Overflow,
       Memory_Violation,
       Hardware_Fault,
       Power_Fail);
   type Error_Status_Type   is record
       Error_Code          : Error_Code_Type;
       Length             : Error_Message_Size_Type;
       Failed_Process_Id :  APEX.Processes.Process_Id_Type;
       Failed_Address     : System_Address_Type;
       Message            : Error_Message_Type;
   end record;
   procedure Report_Application_Message
      (Message_Addr : in Message_Addr_Type;
       Length       : in Message_Size_Type;
       Return_Code : out Return_Code_Type);
   procedure Create_Error_Handler
      (Entry_Point  : in System_Address_Type;
       Stack_Size : in APEX.Processes.Stack_Size_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Error_Status
      (Error_Status : out Error_Status_Type;
       Return_Code : out Return_Code_Type);
   procedure Raise_Application_Error
      (Error_Code   : in Error_Code_Type;
       Message_Addr : in Message_Addr_Type;
       Length       : in Error_Message_Size_Type;
       Return_Code : out Return_Code_Type);
private
   pragma Convention (C, Error_Code_Type);
   pragma Convention (C, Error_Status_Type);
end APEX.Health_Monitoring;
```

8.3.6 Module schedules

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- MODULE_SCHEDULES constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
package APEX.Module_Schedules is
   type Schedule_Id_Type is private;
   Null_Schedule_Id : constant Schedule_Id_Type;
   subtype Schedule_Name_Type is Name_Type;
   type Schedule_Status_Type     is record
       Time_Of_Last_Schedule_Switch        : System_Time_Type;
       Current_Schedule                    : Schedule_Id_Type;
       Next_Schedule                : Schedule_Id_Type;
   end record;
   procedure Set_Module_Schedule
     (Schedule_Id      : in Schedule_Id_Type;
      Return_Code      : out Return_Code_Type);
   procedure Get_Module_Schedule_Status
     (Schedule_Status      : out Schedule_Status_Type;
      Return_Code          : out Return_Code_Type);
   procedure Get_Module_Schedule_Id
     (Schedule_Name     : in Schedule_Name_Type;
      Schedule_Id       : out Schedule_Id_Type;
      Return_Code       : out Return_Code_Type);
private
   Type Schedule_Id_Type is new APEX_Integer;
   Null_Schedule_Id : constant Schedule_Id_Type := 0;
   pragma Convention (C, Schedule_Status_Type);
end APEX.Module_Schedules;
```

8.3.7 Partitions

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- PARTITION constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Partitions is
   Max_Number_Of_Partitions : constant := System_Limit_Number_Of_Partitions;
   type Operating_Mode_Type is (Idle, Cold_Start, Warm_Start, Normal);
   type Partition_Id_Type is private;
   Null_Partition_Id : constant Partition_Id_Type;
   type Start_Condition_Type is
     (Normal_Start,
      Partition_Restart,
      Hm_Module_Restart,
      Hm_Partition_Restart);
   type Partition_Status_Type is record
      Period          : System_Time_Type;
      Duration        : System_Time_Type;
      Identifier      : Partition_Id_Type;
      Lock_Level      : APEX.Processes.Lock_Level_Type;
      Operating_Mode : Operating_Mode_Type;
      Start_Condition : Start_Condition_Type;
   end record;
   procedure Get_Partition_Status
     (Partition_Status : out Partition_Status_Type;
      Return_Code      : out Return_Code_Type);
   procedure Set_Partition_Mode
```

```
      (Operating_Mode : in Operating_Mode_Type;
       Return_Code    : out Return_Code_Type);
private
   type Partition_ID_Type is new APEX_Integer;
   Null_Partition_Id : constant Partition_Id_Type := 0;
   pragma Convention (C, Operating_Mode_Type);
   pragma Convention (C, Start_Condition_Type);
   pragma Convention (C, Partition_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Get_Partition_Status, "GET_PARTITION_STATUS");
   pragma Import (C, Set_Partition_Mode, "SET_PARTITION_MODE");
   -- END OF POK BINDINGS
end APEX.Partitions;
```

8.3.8 Processes

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- PROCESS constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
package APEX.Processes is
   Max_Number_Of_Processes : constant := System_Limit_Number_Of_Processes;
   Min_Priority_Value : constant := 0;
   Max_Priority_Value : constant := 249;
   Max_Lock_Level : constant := 32;
   subtype Process_Name_Type is Name_Type;
   type Process_Id_Type is private;
   Null_Process_Id : constant Process_Id_Type;
   subtype Lock_Level_Type is APEX_Integer range 0 .. Max_Lock_Level;
   subtype Stack_Size_Type is APEX_Unsigned;
   subtype Waiting_Range_Type is APEX_Integer range
      0 .. Max_Number_Of_Processes;
   subtype Priority_Type is APEX_Integer range
      Min_Priority_Value .. Max_Priority_Value;
   type Process_State_Type is (Dormant, Ready, Running, Waiting);
   type Deadline_Type is (Soft, Hard);
   type Process_Attribute_Type is record
       Period        : System_Time_Type;
       Time_Capacity : System_Time_Type;
       Entry_Point   : System_Address_Type;
       Stack_Size    : Stack_Size_Type;
       Base_Priority : Priority_Type;
```

```
        Deadline      : Deadline_Type;
        Name          : Process_Name_Type;
   end record;
   type Process_Status_Type is record
        Deadline_Time    : System_Time_Type;
        Current_Priority : Priority_Type;
        Process_State    : Process_State_Type;
        Attributes       : Process_Attribute_Type;
   end record;
   procedure Create_Process
       (Attributes : in Process_Attribute_Type;
        Process_Id : out Process_Id_Type;
        Return_Code : out Return_Code_Type);
   procedure Set_Priority
       (Process_Id : in Process_Id_Type;
        Priority    : in Priority_Type;
        Return_Code : out Return_Code_Type);
   procedure Suspend_Self
       (Time_Out    : in System_Time_Type;
        Return_Code : out Return_Code_Type);
   procedure Suspend
       (Process_Id : in Process_Id_Type;
        Return_Code : out Return_Code_Type);
   procedure Resume
       (Process_Id : in Process_Id_Type;
        Return_Code : out Return_Code_Type);
   procedure Stop_Self;
   procedure Stop
       (Process_Id : in Process_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Start
      (Process_Id : in Process_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Delayed_Start
      (Process_Id : in Process_Id_Type;
       Delay_Time : in System_Time_Type;
       Return_Code : out Return_Code_Type);
   procedure Lock_Preemption
      (Lock_Level : out Lock_Level_Type;
       Return_Code : out Return_Code_Type);
   procedure Unlock_Preemption
      (Lock_Level : out Lock_Level_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_My_Id
      (Process_Id : out Process_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Process_Id
      (Process_Name : in Process_Name_Type;
       Process_Id    : out Process_Id_Type;
       Return_Code : out Return_Code_Type);
   procedure Get_Process_Status
      (Process_Id      : in Process_Id_Type;
       Process_Status : out Process_Status_Type;
       Return_Code     : out Return_Code_Type);
private
   type Process_ID_Type is new APEX_Integer;
   Null_Process_Id : constant Process_Id_Type := 0;
   pragma Convention (C, Process_State_Type);
   pragma Convention (C, Deadline_Type);
   pragma Convention (C, Process_Attribute_Type);
   pragma Convention (C, Process_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Process, "CREATE_PROCESS");
   pragma Import (C, Set_Priority, "SET_PRIORITY");
   pragma Import (C, Suspend_Self, "SUSPEND_SELF");
   pragma Import (C, Suspend, "SUSPEND");
   pragma Import (C, Resume, "SUSPEND");
   pragma Import (C, Stop_Self, "STOP_SELF");
   pragma Import (C, Stop, "STOP");
   pragma Import (C, Start, "START");
   pragma Import (C, Delayed_Start, "DELAYED_START");
   pragma Import (C, Lock_Preemption, "LOCK_PREEMPTION");
   pragma Import (C, Unlock_Preemption, "UNLOCK_PREEMPTION");
   pragma Import (C, Get_My_Id, "GET_MY_ID");
   pragma Import (C, Get_Process_Id, "GET_PROCESS_ID");
   pragma Import (C, Get_Process_Status, "GET_PROCESS_STATUS");
   -- END OF POK BINDINGS
end APEX.Processes;
```

8.3.9 Queuing ports

```
--                                POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- QUEUING PORT constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Queuing_Ports is
   Max_Number_Of_Queuing_Ports : constant :=
      System_Limit_Number_Of_Queuing_Ports;
   subtype Queuing_Port_Name_Type is Name_Type;
   type Queuing_Port_Id_Type is private;
   Null_Queuing_Port_Id : constant Queuing_Port_Id_Type;
   type Queuing_Port_Status_Type is record
       Nb_Message          : Message_Range_Type;
       Max_Nb_Message      : Message_Range_Type;
       Max_Message_Size : Message_Size_Type;
       Port_Direction      : Port_Direction_Type;
       Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;
   procedure Create_Queuing_Port
      (Queuing_Port_Name : in Queuing_Port_Name_Type;
       Max_Message_Size    : in Message_Size_Type;
       Max_Nb_Message      : in Message_Range_Type;
       Port_Direction      : in Port_Direction_Type;
       Queuing_Discipline : in Queuing_Discipline_Type;
       Queuing_Port_Id     : out Queuing_Port_Id_Type;
       Return_Code         : out Return_Code_Type);
   procedure Send_Queuing_Message
      (Queuing_Port_Id : in Queuing_Port_Id_Type;
       Message_Addr     : in Message_Addr_Type;
       Length           : in Message_Size_Type;
       Time_Out         : in System_Time_Type;
       Return_Code      : out Return_Code_Type);
   procedure Receive_Queuing_Message
      (Queuing_Port_Id : in Queuing_Port_Id_Type;
       Time_Out         : in System_Time_Type;
       Message_Addr     : in Message_Addr_Type;
   -- The message address is passed IN, although the respective message is
   -- passed OUT
       Length           : out Message_Size_Type;
       Return_Code      : out Return_Code_Type);
   procedure Get_Queuing_Port_Id
      (Queuing_Port_Name : in Queuing_Port_Name_Type;
       Queuing_Port_Id    : out Queuing_Port_Id_Type;
       Return_Code        : out Return_Code_Type);
   procedure Get_Queuing_Port_Status
      (Queuing_Port_Id      : in Queuing_Port_Id_Type;
       Queuing_Port_Status : out Queuing_Port_Status_Type;
       Return_Code          : out Return_Code_Type);
private
   type Queuing_Port_Id_Type is new APEX_Integer;
   Null_Queuing_Port_Id : constant Queuing_Port_Id_Type := 0;
   pragma Convention (C, Queuing_Port_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Queuing_Port, "CREATE_QUEUING_PORT");
   pragma Import (C, Send_Queuing_Message, "SEND_QUEUING_PORT_MESSAGE");
   pragma Import (C, Receive_Queuing_Message, "RECEIVE_QUEUING_MESSAGE");
   pragma Import (C, Get_Queuing_Port_Id, "GET_QUEUING_PORT_ID");
   pragma Import (C, Get_Queuing_Port_Status, "GET_QUEUING_PORT_STATUS");
   -- END OF POK BINDINGS
end APEX.Queuing_Ports;
```

8.3.10 Sampling ports

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
```

```
--                                    Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- SAMPLING PORT constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
package APEX.Sampling_Ports is
   Max_Number_Of_Sampling_Ports : constant :=
      System_Limit_Number_Of_Sampling_Ports;
   subtype Sampling_Port_Name_Type is Name_Type;
   type Sampling_Port_Id_Type is private;
   Null_Sampling_Port_Id : constant Sampling_Port_Id_Type;
   type Validity_Type is (Invalid, Valid);
   type Sampling_Port_Status_Type is record
       Refresh_Period    : System_Time_Type;
       Max_Message_Size : Message_Size_Type;
       Port_Direction    : Port_Direction_Type;
       Last_Msg_Validity : Validity_Type;
   end record;
   procedure Create_Sampling_Port
      (Sampling_Port_Name : in Sampling_Port_Name_Type;
       Max_Message_Size   : in Message_Size_Type;
       Port_Direction     : in Port_Direction_Type;
       Refresh_Period     : in System_Time_Type;
       Sampling_Port_Id   : out Sampling_Port_Id_Type;
       Return_Code        : out Return_Code_Type);
   procedure Write_Sampling_Message
      (Sampling_Port_Id : in Sampling_Port_Id_Type;
       Message_Addr     : in Message_Addr_Type;
       Length           : in Message_Size_Type;
       Return_Code      : out Return_Code_Type);
   procedure Read_Sampling_Message
      (Sampling_Port_Id : in Sampling_Port_Id_Type;
       Message_Addr     : in Message_Addr_Type;
   -- The message address is passed IN, although the respective message is
   -- passed OUT
       Length           : out Message_Size_Type;
       Validity         : out Validity_Type;
       Return_Code      : out Return_Code_Type);
   procedure Get_Sampling_Port_Id
      (Sampling_Port_Name : in Sampling_Port_Name_Type;
       Sampling_Port_Id   : out Sampling_Port_Id_Type;
       Return_Code        : out Return_Code_Type);
   procedure Get_Sampling_Port_Status
      (Sampling_Port_Id     : in Sampling_Port_Id_Type;
       Sampling_Port_Status : out Sampling_Port_Status_Type;
       Return_Code          : out Return_Code_Type);
private
   type Sampling_Port_Id_Type is new APEX_Integer;
   Null_Sampling_Port_Id : constant Sampling_Port_Id_Type := 0;
   pragma Convention (C, Validity_Type);
   pragma Convention (C, Sampling_Port_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Sampling_Port, "CREATE_SAMPLING_PORT");
   pragma Import (C, Write_Sampling_Message, "WRITE_SAMPLING_MESSAGE");
   pragma Import (C, Read_Sampling_Message, "READ_SAMPLING_MESSAGE");
   pragma Import (C, Get_Sampling_Port_Id, "GET_SAMPLING_PORT_ID");
   pragma Import (C, Get_Sampling_Port_Status, "GET_SAMPLING_PORT_STATUS");
   -- END OF POK BINDINGS
end APEX.Sampling_Ports;
```

8.3.11 Semaphores

```
--                               POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- SEMAPHORE constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
with APEX.Processes;
package APEX.Semaphores is
   Max_Number_Of_Semaphores : constant := System_Limit_Number_Of_Semaphores;
   Max_Semaphore_Value : constant := 32_767;
   subtype Semaphore_Name_Type is Name_Type;
   type Semaphore_Id_Type is private;
   Null_Semaphore_Id : constant Semaphore_Id_Type;
   type Semaphore_Value_Type is new APEX_Integer range
       0 .. Max_Semaphore_Value;
   type Semaphore_Status_Type is record
       Current_Value      : Semaphore_Value_Type;
       Maximum_Value      : Semaphore_Value_Type;
       Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;
   procedure Create_Semaphore
      (Semaphore_Name      : in Semaphore_Name_Type;
       Current_Value       : in Semaphore_Value_Type;
       Maximum_Value       : in Semaphore_Value_Type;
       Queuing_Discipline : in Queuing_Discipline_Type;
       Semaphore_Id        : out Semaphore_Id_Type;
       Return_Code         : out Return_Code_Type);
   procedure Wait_Semaphore
      (Semaphore_Id : in Semaphore_Id_Type;
       Time_Out      : in System_Time_Type;
       Return_Code  : out Return_Code_Type);
   procedure Signal_Semaphore
      (Semaphore_Id : in Semaphore_Id_Type;
       Return_Code  : out Return_Code_Type);
   procedure Get_Semaphore_Id
      (Semaphore_Name : in Semaphore_Name_Type;
       Semaphore_Id    : out Semaphore_Id_Type;
       Return_Code      : out Return_Code_Type);
   procedure Get_Semaphore_Status
      (Semaphore_Id       : in Semaphore_Id_Type;
       Semaphore_Status : out Semaphore_Status_Type;
       Return_Code        : out Return_Code_Type);
private
   type Semaphore_Id_Type is new APEX_Integer;
   Null_Semaphore_Id : constant Semaphore_Id_Type := 0;
   pragma Convention (C, Semaphore_Status_Type);

   -- POK BINDINGS
   pragma Import (C, Create_Semaphore, "CREATE_SEMAPHORE");
   pragma Import (C, Wait_Semaphore, "WAIT_SEMAPHORE");
   pragma Import (C, Signal_Semaphore, "SIGNAL_SEMAPHORE");
   pragma Import (C, Get_Semaphore_Id, "GET_SEMAPHORE_ID");
   pragma Import (C, Get_Semaphore_Status, "GET_SEMAPHORE_STATUS");
   -- END OF POK BINDINGS
end APEX.Semaphores;
```

8.3.12 Timing

```
--                              POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

-- ---------------------------------------------------------------------------
-- --
-- TIME constant and type definitions and management services --
-- --
-- ---------------------------------------------------------------------------
package APEX.Timing is
   procedure Timed_Wait
      (Delay_Time : in System_Time_Type;
       Return_Code : out Return_Code_Type);
   procedure Periodic_Wait (Return_Code : out Return_Code_Type);
   procedure Get_Time
      (System_Time : out System_Time_Type;
       Return_Code : out Return_Code_Type);
   procedure Replenish
      (Budget_Time : in System_Time_Type;
       Return_Code : out Return_Code_Type);

   -- POK BINDINGS
   pragma Import (C, Timed_Wait, "TIMED_WAIT");
   pragma Import (C, Periodic_Wait, "PERIODIC_WAIT");
   pragma Import (C, Get_Time, "GET_TIME");
   pragma Import (C, Replenish, "REPLENISH");
   -- END OF POK BINDINGS
```

```
end Apex.Timing;
```

Chapter 9

Instrumentation

You can automatically instrument POK using the `--with-instrumentation` option when you configure the build-system (see section 3.3 for more information. In consequence, when you use this mode, more output is produced and additional files are automatically created when the system stops. This section details the files that are automatically produced in this mode and how to use them.

9.1 Instrumentation purpose

At this time, the instrumentation functionnality was done to observe scheduling of partitions and tasks in the Cheddar scheduling analysis tool. If you want to use this functionnality, you have to install Cheddar (see 10.2 for information about Cheddar).

9.2 Output files

When you run your system using the `make run` command and if the instrumentation configuration flag was set, the following files are automatically produced:

- `cheddar-archi.xml`: contains the architecture used with the POK kernel.
- `cheddar-events.xml`: contains the scheduling events that are registered during POK execution.

9.3 Use cheddar with produces files

Start Cheddar. Then, load the XML file `cheddar-archi.xml`. You can also use Cheddar with this file as an argument.

Then, load the `cheddar-events.xml` file. For that, use the menu `Tool`/`Scheduling`/`Event Table Service`/`Import` and choose the generated `cheddar-events.xml` file.

Then, to draw the scheduling diagram, choose the menu option `Tool`/`Scheduling`/`Event Table Service`/`Draw Time Line`.

Cheddar will directly draw the scheduling diagram that corresponds to the execution.

Chapter 10

Annexes

10.1 Terms

- **AADL**: AADL stands for Architecture Analysis and Design Language. It provides modeling facilities to represent a system with their properties and requirements.
- **Leon3**: A processor architecture developped by the European Space Agency.
- **Ocarina**: AADL compiler developed by TELECOM ParisTech. It is used by the POK project to automatically generate configuration, deployment and application code.
- **PowerPC**: Architecture popular in the embedded domain.
- **QEMU**: A general-purpose emulator that runs on various platforms and emulates different processors (such as INTELx86 or PowerPC).

10.2 Resources

- POK website: *http://pok.gunnm.org*
- Ocarina website: *http://aadl.telecom-paristech.fr*
- QEMU website: *http://www.qemu.com*
- Cheddar: *http://beru.univ-brest.fr/ singhoff/cheddar/*
- MacPorts: *http://www.macports.org*

10.3 POK property set for the AADL

```
property set POK is
  Security_Level : aadlinteger applies to
         (virtual processor, virtual bus, process, bus, event data port, event port, data port);
  -- Means two things :
  --   * security_level that a partition is allowed to access
  --   * security_level provided by a virtual bus : ensure that
  --     the virtual bus can transport data from and/or to partitions
  --     that have this security level.

  Criticality : aadlinteger applies to
         (virtual processor);
  --  Represent the criticality level of a partition.

  Handler : aadlstring applies to
         (virtual processor);
  -- Error handler for each partition
  -- By default, the code generator can create a function
  -- which name derives from the partition name. Instead, the
  -- model can provide the name of the handler with this property.

  Topics : list of aadlstring applies to
         (virtual processor, virtual bus);
  -- Means two things :
  --   * The topics allowed on a specific virtual processor
  --   * Topics allowed on a virtual bus.

  Needed_Memory_Size : Size applies to (process);
  -- Specify the amount of memory needed for a partition
  -- We apply it to process component because we don't
  -- isolate virtual processor, only processes

  Available_Schedulers : type enumeration
  (
      RMS,
      EDF,
      LLF,
      RR,
      TIMESLICE,
      STATIC
  );

  Timeslice : Time applies to (virtual processor);
  -- DEPRECATED at this time

  Major_Frame : Time applies to (processor);

  Scheduler : POK::Available_Schedulers
      applies to (processor, virtual processor);

  Slots: list of Time applies to (processor);

  Slots_Allocation:  list of reference (virtual processor) applies to (processor);
  --  List available schedulers
  --  When we use the STATIC scheduler in the virtual processor
  --  The Slots and Slots_Allocation properties are used to determine when
  --  partitions are activated and the timeslice they have for their execution.

  Supported_Error_Code: type enumeration (Deadline_Missed, Application_Error, Numeric_Error, Illegal

  Recovery_Errors   : list of POK::Supported_Error_Code applies to (processor, virtual processor, th

  Supported_Recovery_Action: type enumeration (Ignore, Confirm, Thread_Restart, Thread_Stop_And_Star

  Recovery_Actions : list of POK::Supported_Recovery_Action applies to (processor, virtual processor
  -- There is two properties that handle errors and their recovery at the processor and virtual proc
  -- These two properties must be declared both in the component.
  -- For example, we declare the properties like that:
  --      Recovery_Errors  => (Deadline_Missed, Memory_Violation);
  --      Recovery_Actions => (Ignore, Partition_Restart);
  -- It means that if we have a missed deadline, we ignore the error. But if we get
  -- a memory violation error, we restart the partition.

  Available_BSP : type enumeration
  (
      x86_qemu,
      x86_qemu_vmm,
      prep,
      leon3
  );

  BSP : POK::Available_BSP applies to (processor, system);

  Available_Architectures : type enumeration
  (
      x86, ppc, sparc
  );

  Architecture : POK::Available_Architectures applies to (processor, system);

  -- Deployment properties
  -- Indicate which architecture we use and which bsp

  Source_Location : aadlstring applies to (subprogram);

  -- Indicate where is the object file
  -- that contains this subprogram.


  MILS_Verified : aadlboolean applies to (system, process, device, thread, processor, data);
  --  For verification purpose

  Refresh_Time : Time applies to (data port);

  Hw_Addr : aadlstring applies to (device);

  PCI_Vendor_Id : aadlstring applies to (device);

  PCI_Device_ID : aadlstring applies to (device);

  Device_Name : aadlstring applies to (device);

  Additional_Features : list of POK::Supported_Additional_Features applies to (virtual processor, pr

  Supported_Additional_Features: type enumeration (libmath, libc_stdlib, libc_stdio, libc_string, io

  Des_Key : aadlstring applies to (virtual bus);

  Des_Init : aadlstring applies to (virtual bus);

  Blowfish_Key : aadlstring applies to (virtual bus);

  Blowfish_Init : aadlstring applies to (virtual bus);

  Supported_POK_Protocols: type enumeration (ceasar,des,blowfish,unknown);

  Protocol : POK::Supported_POK_Protocols applies to (virtual bus);
end POK;
```

10.4 AADL library

This is not C code but an AADL library that can be used with your own models. When you use this library, you don't have to specify all your components and properties, just use predefined components to generate your application.

```
--
--                                POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                       Copyright (c) 2007-2022 POK team

package poklib

public

with POK;
with Data_Model;

------------------
--  Processor   --
------------------

processor pok_kernel
properties
   POK::Scheduler => static;
end pok_kernel;

processor implementation pok_kernel.x86_qemu
properties
   POK::Scheduler => static;
   POK::Architecture => x86;
   POK::BSP => x86_qemu;
end pok_kernel.x86_qemu;

processor implementation pok_kernel.ppc_prep
properties
   POK::Architecture => ppc;
   POK::BSP => prep;
end pok_kernel.ppc_prep;

processor implementation pok_kernel.sparc_leon3
properties
   POK::Architecture => sparc;
   POK::BSP => leon3;
end pok_kernel.sparc_leon3;

processor implementation pok_kernel.x86_qemu_two_partitions extends pok_kernel.x86_qemu
subcomponents
   partition1 : virtual processor poklib::pok_partition.basic_for_example;
   partition2 : virtual processor poklib::pok_partition.basic_for_example;
properties
   POK::Major_Frame => 1000ms;
   POK::Scheduler => static;
   POK::Slots => (500ms, 500ms);
   POK::Slots_Allocation => ( reference (partition1), reference (partition2));
end pok_kernel.x86_qemu_two_partitions;

processor implementation pok_kernel.x86_qemu_three_partitions extends pok_kernel.x86_qemu
subcomponents
   partition1 : virtual processor poklib::pok_partition.basic_for_example;
   partition2 : virtual processor poklib::pok_partition.basic_for_example;
   partition3 : virtual processor poklib::pok_partition.basic_for_example;
properties
   POK::Major_Frame => 1500ms;
   POK::Scheduler => static;
   POK::Slots => (500ms, 500ms, 500ms);
   POK::Slots_Allocation => ( reference (partition1), reference (partition2), reference (partition3)
end pok_kernel.x86_qemu_three_partitions;

processor implementation pok_kernel.x86_qemu_four_partitions extends pok_kernel.x86_qemu
subcomponents
   partition1 : virtual processor poklib::pok_partition.basic_for_example;
   partition2 : virtual processor poklib::pok_partition.basic_for_example;
   partition3 : virtual processor poklib::pok_partition.basic_for_example;
   partition4 : virtual processor poklib::pok_partition.basic_for_example;
properties
   POK::Major_Frame => 2000ms;
   POK::Scheduler => static;
   POK::Slots => (500ms, 500ms, 500ms, 500ms);
   POK::Slots_Allocation => ( reference (partition1), reference (partition2), reference (partition3)
end pok_kernel.x86_qemu_four_partitions;

processor implementation pok_kernel.x86_qemu_four_partitions_with_libmath extends pok_kernel.x86_qem
subcomponents
   partition1 : virtual processor poklib::pok_partition.basic_for_example_with_libmath;
   partition2 : virtual processor poklib::pok_partition.basic_for_example_with_libmath;
   partition3 : virtual processor poklib::pok_partition.basic_for_example_with_libmath;
   partition4 : virtual processor poklib::pok_partition.basic_for_example_with_libmath;
```

```
properties
   POK::Major_Frame => 2000ms;
   POK::Scheduler => static;
   POK::Slots => (500ms, 500ms, 500ms, 500ms);
   POK::Slots_Allocation => ( reference (partition1), reference (partition2), reference (partition3)
end pok_kernel.x86_qemu_four_partitions_with_libmath;

---------------------
--  Virtual Buses  --
---------------------

--  Unclassified virtual bus

virtual bus unencrypted
end unencrypted;

virtual bus implementation unencrypted.i
properties
   POK::Protocol        => unknown;
end unencrypted.i;

--  blowfish virtual bus

subprogram blowfish_send
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
properties
   Source_Name => "pok_protocols_blowfish_marshall";
end blowfish_send;

subprogram implementation blowfish_send.i
end blowfish_send.i;

subprogram blowfish_receive
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
properties
   Source_Name => "pok_protocols_blowfish_unmarshall";
end blowfish_receive;

subprogram implementation blowfish_receive.i
end blowfish_receive.i;

data blowfish_data
end blowfish_data;

data implementation blowfish_data.i
properties
   Type_Source_Name => "pok_protocols_blowfish_data_t";
end blowfish_data.i;

```

```
abstract vbus_blowfish_wrapper
end vbus_blowfish_wrapper;

abstract implementation vbus_blowfish_wrapper.i
subcomponents
   send               : subprogram blowfish_send.i;
   receive            : subprogram blowfish_receive.i;
   marshalling_type   : data blowfish_data.i;
end vbus_blowfish_wrapper.i;

virtual bus blowfish
end blowfish;

virtual bus implementation blowfish.i
properties
   Implemented_As           => classifier (poklib::vbus_blowfish_wrapper.i);
   POK::Protocol            => blowfish;
end blowfish.i;

--  DES virtual bus

subprogram des_send
features
   datain      : in parameter    poklib::pointed_void;
   count_in    : in parameter    poklib::integer;
   dataout     : out parameter   poklib::pointed_void;
   count_out   : out parameter   poklib::integer;
end des_send;

subprogram implementation des_send.i
properties
   Source_Name => "pok_protocols_des_marshall";
end des_send.i;

subprogram des_receive
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
end des_receive;

subprogram implementation des_receive.i
properties
   Source_Name => "pok_protocols_des_unmarshall";
end des_receive.i;

data des_data
end des_data;

data implementation des_data.i
properties
   Type_Source_Name => "pok_protocols_des_data_t";
end des_data.i;

abstract vbus_des_wrapper
end vbus_des_wrapper;
```

```

abstract implementation vbus_des_wrapper.i
subcomponents
   send              : subprogram des_send.i;
   receive           : subprogram des_receive.i;
   marshalling_type  : data des_data.i;
end vbus_des_wrapper.i;

virtual bus des
end des;

virtual bus implementation des.i
properties
   Implemented_As          => classifier (poklib::vbus_des_wrapper.i);
   POK::Protocol           => des;
end des.i;

--  ceasar virtual bus

subprogram ceasar_send
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
properties
   Source_Name => "pok_protocols_ceasar_marshall";
end ceasar_send;

subprogram implementation ceasar_send.i
end ceasar_send.i;

subprogram ceasar_receive
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
properties
   Source_Name => "pok_protocols_ceasar_unmarshall";
end ceasar_receive;

subprogram implementation ceasar_receive.i
end ceasar_receive.i;

data ceasar_data
end ceasar_data;

data implementation ceasar_data.i
properties
   Type_Source_Name => "pok_protocols_ceasar_data_t";
end ceasar_data.i;


abstract vbus_ceasar_wrapper
end vbus_ceasar_wrapper;

```

```
abstract implementation vbus_ceasar_wrapper.i
subcomponents
   send              : subprogram ceasar_send.i;
   receive           : subprogram ceasar_receive.i;
end vbus_ceasar_wrapper.i;

virtual bus ceasar
end ceasar;

virtual bus implementation ceasar.i
properties
   Implemented_As          => classifier (poklib::vbus_ceasar_wrapper.i);
   POK::Protocol           => ceasar;
end ceasar.i;

--  gzip virtual bus

subprogram gzip_send
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
properties
   Source_Name => "pok_protocols_gzip_marshall";
end gzip_send;

subprogram implementation gzip_send.i
end gzip_send.i;

subprogram gzip_receive
features
   datain   : in parameter poklib::pointed_void;
   countin  : in parameter poklib::integer;
   dataout  : out parameter poklib::pointed_void;
   countout : out parameter poklib::integer;
properties
   Source_Name => "pok_protocols_gzip_unmarshall";
end gzip_receive;

subprogram implementation gzip_receive.i
end gzip_receive.i;

data gzip_data
end gzip_data;

data implementation gzip_data.i
properties
   Type_Source_Name => "pok_protocols_gzip_data_t";
end gzip_data.i;

abstract vbus_gzip_wrapper
end vbus_gzip_wrapper;

abstract implementation vbus_gzip_wrapper.i
subcomponents
   send              : subprogram gzip_send.i;
   receive           : subprogram gzip_receive.i;
   marshalling_type  : data gzip_data.i;
end vbus_gzip_wrapper.i;

virtual bus gzip
end gzip;

virtual bus implementation gzip.i
properties
   Implemented_As          => classifier (poklib::vbus_gzip_wrapper.i);
end gzip.i;

-------------------------
--  Virtual Processor  --
-------------------------

virtual processor pok_partition
end pok_partition;

virtual processor implementation pok_partition.basic
properties
   POK::Scheduler => RR;
end pok_partition.basic;

virtual processor implementation pok_partition.driver
properties
   POK::Scheduler => RR;
   POK::Additional_Features => (pci, io);
end pok_partition.driver;

virtual processor implementation pok_partition.application
properties
   POK::Scheduler => RR;
   POK::Additional_Features => (libc_stdio, libc_stdlib);
end pok_partition.application;

virtual processor implementation pok_partition.basic_for_example extends pok_partition.basic
properties
   POK::Additional_Features => (libc_stdio, libc_stdlib);
end pok_partition.basic_for_example;

virtual processor implementation pok_partition.basic_for_example_with_libmath extends pok_partition.
properties
   POK::Additional_Features => (libc_stdio, libc_stdlib, libmath, console);
end pok_partition.basic_for_example_with_libmath;


----------------
--  Memories  --
----------------

memory pok_memory
end pok_memory;

memory implementation pok_memory.x86_segment
end pok_memory.x86_segment;

```

```
memory implementation pok_memory.x86_main
end pok_memory.x86_main;


---------------
--  Threads  --
---------------

thread thr_periodic
properties
   Dispatch_Protocol          => Periodic;
   Period                     => 100ms;
   Deadline                   => 100ms;
   Compute_Execution_Time     => 5ms .. 10ms;
end thr_periodic;

thread thr_sporadic
properties
   Dispatch_Protocol          => Sporadic;
   Period                     => 100ms;
   Deadline                   => 100ms;
   Compute_Execution_Time     => 5ms .. 10ms;
end thr_sporadic;

-------------------
--  Subprograms  --
-------------------

subprogram spg_c
properties
   Source_Language  => C;
   Source_Text      => ("../../../user-functions.o");
end spg_c;

---------------
--  Integer  --
---------------

data void
properties
   Type_Source_Name => "void";
end void;

data implementation void.i
end void.i;

data pointed_void
properties
   Type_Source_Name => "void*";
end pointed_void;

data implementation pointed_void.i
end pointed_void.i;

data char
properties
   Type_Source_Name => "char";
```

```
end char;

data implementation char.i
end char.i;

data pointed_char
properties
   Type_Source_Name => "char*";
end pointed_char;

data implementation pointed_char.i
end pointed_char.i;

data integer
properties
   Data_Model::Data_Representation => integer;
end integer;


data float
properties
   Data_Model::Data_Representation => float;
end float;




end poklib;
```

10.5 ARINC653 property set for the AADL

10.6 Network example, modeling of device drivers

```
--
--                                 POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                      Copyright (c) 2007-2022 POK team

package rtl8029

--  Be careful when you modify this file, it is used
--  in the annexes of the documentation

public
with POK;
with types;
```

```

data anydata
end anydata;

subprogram init
properties
   source_name => "rtl8029_init";
   source_language => C;
end init;

subprogram poll
properties
   source_name => "rtl8029_polling";
   source_language => C;
end poll;


thread driver_rtl8029_thread
features
   outgoing_topsecret      : out data port types::integer;
   incoming_topsecret      : in data port types::integer;
   outgoing_secret         : out data port types::integer;
   incoming_secret         : in data port types::integer;
   outgoing_unclassified   : out data port types::integer;
   incoming_unclassified   : in data port types::integer;
properties
   Dispatch_Protocol => Periodic;
   Compute_Execution_Time => 0 ms .. 1 ms;
   Period => 1000 Ms;
end driver_rtl8029_thread;

thread driver_rtl8029_thread_poller
properties
   Dispatch_Protocol => Periodic;
   Compute_Execution_Time => 0 ms .. 1 ms;
   Period => 100 Ms;
end driver_rtl8029_thread_poller;

thread implementation driver_rtl8029_thread.i
connections
   port incoming_unclassified -> outgoing_unclassified;
   port incoming_secret -> outgoing_secret;
   port incoming_topsecret -> outgoing_topsecret;
end driver_rtl8029_thread.i;

thread implementation driver_rtl8029_thread_poller.i
calls
   call1 : { pspg : subprogram poll;};
end driver_rtl8029_thread_poller.i;

process driver_rtl8029_process
end driver_rtl8029_process;

process implementation driver_rtl8029_process.i
subcomponents
   thr : thread driver_rtl8029_thread.i;
   poller   : thread driver_rtl8029_thread_poller.i;
   properties
     POK::Needed_Memory_Size => 160 Kbyte;
   end driver_rtl8029_process.i;


   abstract driver_rtl8029
   end driver_rtl8029;

   abstract implementation driver_rtl8029.i
   subcomponents
      p : process driver_rtl8029_process.i;
   end driver_rtl8029.i;

end rtl8029;
```

```
--
--                                 POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
--
--                                          Copyright (c) 2007-2022 POK team

package runtime
public
   with POK;
   with layers;
   with types;

   virtual processor partition
   properties
      POK::Scheduler => RR;
      POK::Additional_Features => (libc_stdio, console);
   end partition;

   virtual processor implementation partition.i
   end partition.i;

   processor pok_kernel
   properties
      POK::Architecture => x86;
      POK::BSP => x86_qemu;
   end pok_kernel;

   device separation_netif
   features
      the_bus                 : requires bus access separation_bus.i;
      outgoing_topsecret      : out data port types::integer;
      incoming_topsecret      : in data port types::integer;
      outgoing_secret         : out data port types::integer;
      incoming_secret         : in data port types::integer;
      outgoing_unclassified   : out data port types::integer;
      incoming_unclassified   : in data port types::integer;
   properties
      Initialize_Entrypoint => classifier (rtl8029::init);
      POK::Device_Name => "rtl8029";
   end separation_netif;

   device implementation separation_netif.i
   end separation_netif.i;

   processor implementation pok_kernel.impl
   subcomponents
      runtime_secret       : virtual processor partition.i
                  {
                     Provided_Virtual_Bus_Class => (classifier (layers::secret));
                     POK::Criticality => 10;
                  };
      runtime_topsecret    : virtual processor partition.i
                  {  Provided_Virtual_Bus_Class => (classifier (layers::top_secret));
                     POK::Criticality => 5;
                  };
      runtime_unclassified : virtual processor partition.i
                  {
                     Provided_Virtual_Bus_Class => (classifier (layers::unclassified));
                     POK::Criticality => 1;
                  };
      runtime_netif : virtual processor partition.i
                  {Provided_Virtual_Bus_Class => (classifier (layers::unclassified), classifier (lay
                   POK::Additional_Features => (libc_stdlib, pci, io);
                   POK::Criticality => 10;};
   properties
      POK::Major_Frame => 2000ms;
      POK::Scheduler => static;
      POK::Slots => (500ms, 500ms, 500ms, 500ms);
      POK::Slots_Allocation => ( reference (runtime_secret), reference (runtime_topsecret), referenc
   end pok_kernel.impl;

   bus separation_bus
   end separation_bus;

   bus implementation separation_bus.i
   subcomponents
      layer_topsecret      : virtual bus layers::top_secret;
      layer_secret         : virtual bus layers::secret;
      layer_unclassified   : virtual bus layers::unclassified;
   end separation_bus.i;
end runtime;
```

```
--
--                                POK header
--
-- The following file is a part of the POK project. Any modification should
-- be made according to the POK licence. You CANNOT use this file or a part
-- of a file for your own project.
--
-- For more information on the POK licence, please see our LICENCE FILE
--
-- Please follow the coding guidelines described in doc/CODING_GUIDELINES
```

```
--
--                                     Copyright (c) 2007-2022 POK team

package model
public
   with runtime;
   with partitions;
   with memories;
   with POK;

   system main
   end main;

   system implementation main.i
   subcomponents
      node1_partition_topsecret        : process partitions::process_sender.i;
      node1_partition_secret           : process partitions::process_sender.i;
      node1_partition_unclassified     : process partitions::process_sender.i;
      node2_partition_topsecret        : process partitions::process_receiver.i;
      node2_partition_secret           : process partitions::process_receiver.i;
      node2_partition_unclassified     : process partitions::process_receiver.i;
      node1_memory                     : memory memories::main_memory.i;
      node2_memory                     : memory memories::main_memory.i;
      node1_netif                      : device runtime::separation_netif.i
                                          {POK::Hw_Addr => "00:1F:C6:BF:74:06";
                                           Implemented_As => classifier (rtl8029::driver_rtl8029.i);
                                          };
      node2_netif                      : device runtime::separation_netif.i
                                          {POK::Hw_Addr => "00:0F:FE:5F:7B:2F";
                                           Implemented_As => classifier (rtl8029::driver_rtl8029.i);
                                          };
      node1                            : processor runtime::pok_kernel.impl;
      node2                            : processor runtime::pok_kernel.impl;
      rtbus                            : bus runtime::separation_bus.i;
   connections
      -- Ports of partitions of the first node are connected to the device of this node
      port node1_partition_topsecret.outgoing      -> node1_netif.incoming_topsecret;
      port node1_partition_secret.outgoing         -> node1_netif.incoming_secret;
      port node1_partition_unclassified.outgoing   -> node1_netif.incoming_unclassified;

      -- Ports of partitions of the second node are connected to the device of this second node
      port node2_netif.outgoing_topsecret -> node2_partition_topsecret.incoming;
      port node2_netif.outgoing_secret -> node2_partition_secret.incoming;
      port node2_netif.outgoing_unclassified -> node2_partition_unclassified.incoming;

      port node1_netif.outgoing_topsecret    -> node2_netif.incoming_topsecret
         {Actual_Connection_Binding => (reference (rtbus.layer_topsecret));};
      port node1_netif.outgoing_secret    -> node2_netif.incoming_secret
         {Actual_Connection_Binding => (reference (rtbus.layer_secret));};
      port node1_netif.outgoing_unclassified    -> node2_netif.incoming_unclassified
         {Actual_Connection_Binding => (reference (rtbus.layer_unclassified));};
      bus access rtbus -> node1_netif.the_bus;
      bus access rtbus -> node2_netif.the_bus;
   properties
      Actual_Processor_Binding   => (reference (node1.runtime_topsecret))
applies to node1_partition_topsecret;
      Actual_Processor_Binding   => (reference (node1.runtime_secret))
```

```
applies to node1_partition_secret;
      Actual_Processor_Binding   => (reference (node1.runtime_unclassified)) applies to node1_partit
      Actual_Processor_Binding   => (reference (node2.runtime_topsecret))
applies to node2_partition_topsecret;
      Actual_Processor_Binding   => (reference (node2.runtime_secret))
applies to node2_partition_secret;
      Actual_Processor_Binding   => (reference (node2.runtime_unclassified)) applies to node2_partit
      Actual_Processor_Binding   => (reference (node1.runtime_netif))
applies to node1_netif;
      Actual_Processor_Binding   => (reference (node2.runtime_netif))
applies to node2_netif;
      Actual_Memory_Binding      => (reference (node1_memory.topsecret))
applies to node1_partition_topsecret;
      Actual_Memory_Binding      => (reference (node1_memory.secret))
applies to node1_partition_secret;
      Actual_Memory_Binding      => (reference (node1_memory.unclassified))
applies to node1_partition_unclassified;
      Actual_Memory_Binding      => (reference (node2_memory.topsecret))
applies to node2_partition_topsecret;
      Actual_Memory_Binding      => (reference (node2_memory.secret))
applies to node2_partition_secret;
      Actual_Memory_Binding      => (reference (node2_memory.unclassified))
applies to node2_partition_unclassified;
      Actual_Memory_Binding      => (reference (node2_memory.driver))
applies to node2_netif;
      Actual_Memory_Binding      => (reference (node1_memory.driver))
applies to node1_netif;
   end main.i;

end model;
```